Defence Research and Development Canada Recherche et développement pour la défense Canada

DEFENCE DÉFENSE

Long-term operating system maintenance

A Linux case study

R. Carbone
DRDC Valcartier

Defence R&D Canada – Valcartier
Technical Note
DRDC Valcartier TN 2007-150
January 2008

Canada

Long-term operating system maintenance

A Linux case study

Richard Carbone
DRDC Valcartier

Defence R&D Canada – Valcartier

Technical Note
DRDC Valcartier TN 2007-150
January 2008

Principal Author

Richard Carbone

Programmer/Analyst

Approved by

Stéphan Paradis

Acting Head/Intelligence & Information Section

Norbert Haché (DMSS)

Abstract

In *Operating system hardware reconfiguration: A case study for Linux*, it was determined through experimentation that a Linux-based C2 operating system can successfully undergo a hardware migration and operating system hardware reconfiguration. The direct benefit of this is the ability to forgo any new operating system reinstallation in order to support newer hardware by using mechanisms internal to the operating system that support changes in hardware; this results in a decreased waiting time for system reaccredidation and redeployment. Since an operating system can evolve over time, it can accommodate changes in the system's hardware, thus presenting a tangible advantage for the Navy as this allows the operating system to be maintained over the long-term. However, there are complexities involved when maintaining an operating system for long periods. Therefore, this report serves as an introduction and a simple methodology for performing system maintenance-related tasks that include upgrading, updating, as well as data backups and restoration. This report is neither all-inclusive nor a replacement for qualified system administrators with years of experience. Instead, it can be used as a useful source of information to provide recommended practices, procedures, and information to help in planning for long-term system maintenance.

This page intentionally left blank.

Executive summary

Long-term operating system maintenance: A Linux case study

Carbone, R.; DRDC Valcartier TN 2007-150; Defence R&D Canada – Valcartier; January 2008.

The Canadian Navy's Directorate of Maritime Ship Support (DMSS), under the auspice of the Halifax Modernized Command Control System (HMCCS) project requested that DRDC Valcartier perform an evaluation verifying if the Linux operating system has the ability to withstand various hardware upgrades over its expected lifetime as the new Halifax-class C2 operating system. This has been examined in *Operating system hardware reconfiguration: A case study for Linux*, a study that examined Linux-based hardware migration and operating system hardware reconfiguration. Prior to this study, other work mandated by the Navy requested that DRDC Valcartier examine various long-tem support strategies appropriate for the long-term maintenance of a FOSS-based operating system such as Linux. These findings were laid out in *Life-Cycle Support for Information Systems Based on Free and Open Source Software*.

Once the long-term support strategies and necessary reconfiguration capabilities have been established, an appropriate methodology could be developed for upgrading and updating an operating system in order to ensure a smooth transition to newer hardware via operating system hardware reconfiguration or hardware migration. Therefore, this study proposes a methodology for upgrading and updating an operating system so that installing a newer operating system is not required in order to support newer hardware. Developing a suitable methodology for maintaining an operating system so that it can adapt to hardware changes due to periodic hardware changes was not trivial. Many factors had to be examined and taken into account. It is highly suggested that the reader have some basic knowledge of Linux, UNIX, and system administration before reading this study.

This study should not be construed as authoritative but rather as a guide for performing the necessary system administration related tasks required for system updating and upgrading. Additionally, a section on backup-related issues has been included so that readers can familiarize themselves with it prior to undertaking any system maintenance-related task.

This page intentionally left blank.

Table of contents

1. Introduction

1.1 Objective

The objective of this Technical Note is to examine an update and upgrade-based methodology that can be applied to the Linux operating system in order to maintain it and its hardware throughout its expected long-term service. It is expected that over the years the Navy's frigate C2 systems will not only require operating system maintenance but that they will also periodically undergo hardware upgrades. This Technical Note addresses the necessary methodologies required to proceed with an operating system update or upgrade and thus enable the system to support and accommodate for periodic hardware changes. This is accomplished through two methodologies; the first provides an outline for maintaining system integrity via backups and restorations and the other examines the various issues that must be assessed prior to performing any system-related maintenance. Both methodologies, when combined together form a comprehensive system maintenance methodology that will help the reader succeed in maintaining his system over the long-term.

1.2 Background

1.2.1 Reports

This Technical Note is the fourth and final report in a series of reports prepared for the Navy. The very first report, Report [2], examined the various long-term support strategies needed for maintaining an open source operating system such as Linux. Specifically, it was determined that long-term support could be achieved by (in order of preference) by: 1) the maintainer of the distribution (i.e. vendor); 2) a sub-contractor; 3) the Navy, and 4) a consortium.

The second report, Report [3], was an in-depth analysis of two licenses that compared Red Hat and Novell Suse's enterprise Linux operating systems. They were examined because the Navy had expressed interest in these two operating systems due to: 1) both companies are North American and 2) are already involved with the military establishment, particularly in the U.S.

The third report, Report [1], through experimentation and observation examined the Linux operating system's ability to adapt to changes in its underlying hardware. It was found that Linux can reconfigure itself in order to take advantage of newer hardware, but only if the kernel is recent enough to support that newer hardware. It was concluded that both long-term system maintenance and hardware adaptation could be accomplished in so long as the operating system's kernel is kept up to date.

Reports [5, 6, and 7] are informal internal reports that were the precursor to both this Technical Note and reports [1, 3]. These internal reports were not rigorous enough to be released as formal reports from DRDC.

This Technical Note is the final report in this series. It examines the methodology necessary for maintaining an operating system so that it can be supported over the long-term with the added benefit of successfully accommodating and periodic hardware upgrades.

1.2.2 Mandates

All the reports mandated, including this one have been conducted for the Navy's Directorate of Maritime Ship Support (DMSS 8), under the auspice of the Halifax Modernized Command Control System (HMCCS). These reports have been useful for providing technical information to the Navy aiding it to determine which new operating system, if any, would replace the current C2 operating system aboard the Canadian Halifax-class frigate. These frigates, including both the computer operating system and hardware are currently undergoing modernization.

The original mandate of DRDC Valcartier was to examine and address the various long-term support strategies available to the Navy should they decide to pursue the deployment of a Linux-based operating system aboard the frigates. Furthermore, the Navy wished to understand through which means long-term operating system maintenance could be achieved and supported. This work was carried out in Report [2].

Report [3] was a direct response to a question posed by the HMCCS project that was to conduct a comparison between the corporate licenses of two popular North American Linux vendors. The conclusion that was reached may have an important impact on determining the outcome of which vendor's distribution could potentially be used as the new C2 operating system.

However, as work was underway for reports [2, 3] the HMCCS project requested that DRDC Valcartier examine whether the Linux operating system could adapt to periodic changes in hardware and determine how this would affect overall long-term support. This was studied in Report [1] and consisted of a series of short-lived experiments and observations that was the first of its kind as none could be found in the public literature. Since the new C2 operating system could be in use for 15 to 25 years, the Navy needed to ascertain the maintainability of the operating system and its ability to adapt to periodic hardware changes that are bound to occur.

This Technical Note, although not specifically mandated, was a result of the previous reports that examined the Navy's potential use and implementation of Linux. Specifically, this Technical Note uses results obtained in Report [1] and goes one step further by examining how the Linux operating system can be maintained over the long-term using a vendor-neutral technical methodology. The outcome is a report that provides a global overview of Linux system maintenance that can be understood by system administrators and others with some experience in Linux or UNIX.

1.3 Particulars

1.3.1 Reader

It is both assumed and required that the reader be familiar with Linux, UNIX, or BSD-based operating systems and system administration-related maintenance tasks to better comprehend the various material presented herein. For reasons of brevity, it is not possible to provide all the required background, contextual and technical information (including commands) that a full in-depth examination would normally provide. It is for these reasons that the reader must have at least some knowledge of UNIX and system administration otherwise the reader will find the text to be confusing and cumbersome.

This text is vendor neutral, is silent on many issues (including low-level commands), and assumes that the reader is well versed and understands many of these technical issues. This Technical Note is not a substitute for the reader's experience that is inevitably required in order to complete some of the various tasks examined herein. Finally, it is also assumed that the reader will be using a commercially supported Linux-based operating system.

1.3.2 Navy

The Navy prefers that changes to the operating system be as minimal as possible in order to accelerate the process of reaccredidation and recertification. However, where operating system maintenance is concerned, this is not always that easy. Much will depend on what requires maintenance and how it will be performed. There are many variables to consider prior to performing any system maintenance-related action. Guided by the two methodologies the reader can perform the appropriate set of actions necessary to maintain the system and support the newer hardware.

However, the Navy must understand that operating system maintenance is not always that simple. The main method of system maintenance employs operating system updates, upgrades, and manual system maintenance. Updates and upgrades are generally provided by the distribution's vendor while manual system maintenance is performed by the customer or support provider. Each method has its own advantages and disadvantages. Ultimately, the maintenance option that will be used at any point in time will depend on what requires maintenance as well as who is providing maintenance support.

1.3.3 Report

The Technical Note is broken down into various main topics or sections. The first is the introduction that presents an overview to the reader. The second section examines technical information about operating systems and hardware support. This section, without any previous UNIX experience will be more difficult to understand and contextualize. The third section is a broad overview for performing backup and restoration-related tasks. While this section goes in-depth, it is not meant as a replacement for various books already written on the subject; instead, it presents material that should be considered prior to undertaking any specific task or action. In addition, section does present some new information not present in existing books on the topic. Finally, the fourth section is a broad overview of operating system maintenance that examines the higher-level issues surrounding updates and upgrades. Manual system maintenance has been deliberately excluded from this section, as it is technically complex and cumbersome to examine in a short report. Furthermore, it varies considerably from software package to software package and its success is highly dependent on the experience and knowledge of the reader.

This Technical Note is not meant to be an all-encompassing authority on the various subjects of interest; instead, as a guide it should be used for gaining a higher-level overview on how to approach the subject matter from a global perspective. Certain topics that could have been explored more in-depth were left out to remain vendor neutral.

Furthermore, in order to ascertain this study's relevancy it is important to understand the correlation between the operating system and its life expectancy. As a function of time hardware is expected to change, it is therefore only reasonable to expect that a C2 system will undergo hardware changes. In order for the operating system to support newer hardware, it must be kept up to date. This is particularly important for hardware employing technology that is not currently supported by older kernels and can be accomplished through updates, upgrades, and manual system maintenance.

1.3.4 Methodology

A generic methodology will help to accommodate for the various types of long-term system maintenance and allow the reader to use whichever form is most suitable to his current requirements. However, due to the generalities involved in formulating a generic methodology, many specifics are kept out of the overall process so that the reader can decide which tools and other specifics are most appropriate. Furthermore, a generic approach keeps the text short and concise. In addition, in order to maintain brevity and simplicity, some issues, concepts, and tasks have been altogether left out; they should not affect the overall clarity, as they are minor details that can be filled in by the reader's experience and knowledge.

It is important to understand the difference between an operating system update and upgrade. They differ only in the magnitude of changes made to the operating system. An update generally has less overall impact on the operating system than an upgrade. However, in order to ensure a minimal impact for either they should be performed periodically.

Furthermore, a generic methodology is the best possible approach in order to ensure a successful hardware migration and operating system hardware reconfiguration whenever newer hardware must be supported. As examined in Report [1], a migration and reconfiguration are generally the most appropriate methods for either adapting to newer hardware or moving from one platform to the next. By avoiding the necessity of installing a newer operating system, many potential issues of conflict and contention can altogether be avoided. In addition, employing a generic methodology frees the reader to focus on higher-level issues.

A generic methodology allows for a global overview of the concepts and tasks to be accomplished. How they are completed and through which means is of little concern here. Finally, a generic approach facilitates the interchanging of various lower-level technologies and tools as the higher-level concepts (i.e. the methodology) are generally left unaffected by lower-level changes.

2. Technical background

2.1 Objective

In this section, several important aspects important for operating systems, regardless of the type of system or underlying computer platform are examined. The first aspect presents basic technical information on operating system migrations and reconfigurations. The second presents technical information on operating systems including dependencies and call facilities. Thirdly, compatibility-based issues are briefly examined.

2.2 Reconfigurations and migrations

2.2.1 Background

The most common reason for performing a hardware reconfiguration is to save time and avoid a complete security recertification of the operating system. Rather than install a newer operating system, often considered a daunting task, it may be more appropriate to leverage on the existing operating system with its current configurations and settings.

The Navy has stated that it may not be possible for them to perform full operating system upgrades[1] due to restraints in security, certification, auditing, system and change management as well as overall maintenance. Essentially, the Navy is opting to freeze or "lock out" the operating system with a given configuration for its entire lifetime of between 15 and 25 years. However, it is reasonable to assume that over this period there will be periodic hardware upgrades to the C2 systems. These upgrades may include minor hardware changes or be complete system overhauls.

Generally, outdated operating systems that are not kept at least partially up to date will not fair well in supporting modern hardware. Although the Navy may not be able to switch to a more recent operating system, it can nevertheless benefit from a reconfiguration in so long as the kernel, drivers, and other subsystems are kept to reasonably up to date. Thus, existing applications, preferences, and configurations can be maintained, as can the majority of operating system dependencies and interdependencies, thereby forgoing the necessity for an in-depth recertification of the system before redeployment. Only software that has actually changed would need to be tested and recertified; specifically, the kernel. In large modern operating systems such as Linux, the kernel normally represents less than 1% to 2% of the entire operating system. Furthermore, using modern techniques in static analysis it becomes possible to better assess various vulnerabilities and flaws in the kernel source code [8, 9]. This will help to make a multi-month recertification process last only several weeks, significantly reducing costs and complexities.

[1] By this it is meant going from one version of a Linux distribution to another version of that distribution. An example of this would be going from Red Hat Enterprise Linux (RHEL) 3 to version RHEL 4.

2.2.2 Reconfigurations

A hardware reconfiguration, with respect to reports [1, 5, 6, and 7] is also commonly known as *operating system hardware reconfiguration*, *operating system reconfiguration*, or *reconfiguration*, although they all have the same meaning. They can be collectively defined as "*the ability for an operating system to effectively deal with any changes to the underlying hardware and effectively perpetuate those changes to the appropriate software layers of the operating system such that the changes should remain as transparent as possible to the user*."

Several things must be present for a successful reconfiguration prior to changing any hardware. Firstly, the operating system must actually provide software-based support for hardware detection/redetection. Subsequently, a mechanism must exist by which hardware detection/redetection can be triggered, either automatically at system start-up or manually by the system administrator at any given time. Thirdly, the changes to the operating system must be effectuated in such a manner that they are transparent to the end-user and do not generally require the administrator to manually modify system configuration files.

The majority of today's Linux-based distribution kernels are built "out of the box" to support a wide variety of hardware devices and computing platforms. Most of these operating systems also support various mechanisms for detecting hardware during their initial setup and installation phases. They also have redetection-based tools to detect post-installation changes to the hardware and make the necessary operating system changes to the various operating system configuration files. Some operating systems detect hardware changes automatically, sometimes referred to as *dynamic reconfiguration*, while others do not have this ability and must be run manually, and referred to as *static reconfiguration*. So long as the operating system supports either dynamic or static, it can be said that the operating system is capable of hardware reconfiguration. Whether it is dynamic or static is perhaps a reflection of the level of sophistication of the operating system itself and therefore static and dynamic reconfigurations can be treated as one in the same. However, the ability for a reconfiguration to occur is commensurate with the maturity of the operating system. Older Linux systems often encountered difficulty supporting existing hardware; in such cases where older Linux systems are used, it is not realistic to expect it to support newer changes in hardware.

2.2.3 Migrations

A *hardware migration* (with respect to reports [1, 5, 6, and 7]), often referred to as *migration*, is a term similar to operating system hardware reconfiguration. However, they differ in that a migration[2] is an operating system hardware reconfiguration that takes place only after the entire underlying computing platform has been replaced. In other words, the operating system is altogether transferred to another computer by various means (i.e. disk copying, etc.). An example of this would be moving a Linux-based operating system from a Pentium-class system to a Pentium IV-class system. Most, if not all the new system's hardware will be completely different from that of its predecessor. An operating system hardware reconfiguration then takes place only once the power is applied to the new system and is allowed to boot. Depending on the maturity of the Linux operating system in question, the reconfiguration, if it occurs may be either dynamic or static in nature.

[2] Migrations can only be performed on systems with the same basic architecture (i.e. x86, x64, etc.).

2.3 Operating systems

2.3.1 Background

Operating systems are far larger and complex than they were, even ten years ago. For this reason, in this section, a brief examination of operating systems and the technical issues as well as complexities surrounding them will be examined in the following subsections. Although this section is specifically concerned with the Linux operating system, it is equally applicable to other open source and non-open source operating systems as well.

2.3.2 Definition

The term operating system has been given many varying definitions over the years. In this text, the more classical definition of the term is used. An operating system can be big or small; it is not defined by size. At a minimum, an operating system is a collection of executable code separated into individual computer files that controls the computer's hardware and user-based applications, tools, and utilities. An operating system is compromised of: 1) a kernel; 2) a shell or GUI (for interacting with the kernel and launching applications); and 3) user-based applications.

The kernel provides the low-level facility (or layer) that interacts directly with the system's hardware. The kernel consists of various device drivers, memory management components, and a general framework for interacting with the hardware and running applications. The shell or GUI accepts user-based input used for interacting with the system's hardware and running various applications and utilities that perform some useful work on the user's behalf. Finally, user applications and utilities exist to provide services and functionality to the user.

With Linux, it is important to understand the difference between the Linux kernel and a Linux-based operating system. Linux, in of itself, refers only to the GNU/Linux kernel which includes drivers and various subsystems such as memory management components. A Linux-based operating system, at its most basic, is no more different from any other computer operating system. It too is compromised of a kernel, a shell (or GUI) for user-based interactions with the kernel, and a collection of user-based applications. Thus, a Linux-based operating system is more than just GNU/Linux kernel, but also includes a command line shell or GUI, an assortment of user-based tools, applications, utilities, documentation, source code, etc. Together these form a usable and functional computer operating system.

2.3.3 System dependencies

2.3.3.1 Interdependencies

Every operating system has interdependencies. Interdependency exists where one or more software components rely on other components. These components can be either related (part of the same application) or completely unrelated (from a different application). Related or not, there is nevertheless a relationship between the various components. In this relationship, one component requires something from the other. It could also be that both components are co-dependent (dependent on each other) or the dependency could be one-way only. Most often, it is functions and API's that are required by one component from another. However, depending on how the software was written (including the overall design) co-dependent components do exist where they require one another for their own proper functioning.

To better understand interdependency, it helps to use a simple example. For instance, the example of an application and software library come easily to mind. In order for the application to work correctly, it requires access to certain functions and API's that can only be satisfied by that specific library. Otherwise, the programmer will have to recreate those functions and API's; this is a very complex and time-consuming process. However, often time the software library is not actually a part of the application; instead, it is from a different application altogether (this is particularly common in UNIX). However, because the programmer wishes to reduce the amount and complexity of his work, using functions and API's developed by others reduces his time, work, and expended effort. Programmers tend to prefer building on pre-existing work (i.e. functions, API's, system calls) since doing otherwise can lead to increased programming and operating system complexity.

In large Linux-based operating systems, it is common to have many thousands of such interdependencies. Such a system could have thousands of applications installed and most of them will require functions and API's from any one of the various software libraries found distributed across the operating system. Although it is possible to develop entirely self-contained applications, it is normally not done, as rewriting existing library functionality already available is unproductive.

2.3.3.2 System calls

Another type of interdependency is the system call. A system call, according to [4] is an operating system subroutine that provides a uniform manner for performing operating system-level tasks such as deleting files, managing directories, opening hardware for I/O, communicating with the network, etc. A system call relies on pre-existing functions and API's provided by the kernel. Using system calls a programmer does not have recreate a function or API already provided by the system; furthermore, in all likelihood, the new function would not be as efficiently written. Because the kernel controls the system's hardware, it makes sense to place low-level functionality into the kernel that can be used by developers and programmers alike. When a low-level task (i.e. routine) is called by an application, it places the system call and passes control to the kernel which then runs the appropriate subroutine (i.e. system call), and once completed, returns control back to the requesting application.

2.4 Compatibility issues

2.4.1 Background

Due to its modular nature the Linux, operating system can be replaced in whole or in pieces. Of course, the issue of software and library interdependencies may need to be resolved when updating or upgrading both software and the operating system.

The openness of Linux makes it possible to see the interdependent nature of applications that rely on various software libraries and the kernel. There are tools that come bundled with most Linux systems that enable the user to examine the various functions and system calls made by requesting applications for library and/or kernel functionality. These tools can also be used to examine library-based issues (i.e. inconsistencies and incompatibilities) that can arise when they are replaced by newer versions. Sometimes the problem with newer libraries is that older functionality is removed thus causing issues with older dependent applications and libraries; nevertheless, this functionality may still be required by certain applications. Therefore, these various troubleshooting tools can also be used to track down specifics that may be indicative of a potential failure when upgrading libraries and or the kernel.

When upgrading a kernel or software library, it is not always possible to determine the impact that changes to these libraries or kernel may have on the system. This is why it is important to test applications after upgrades.

2.4.2 Upgrades

Upgrades impose newer versions of software on the system. Due to the modular nature of Linux this is generally not an issue. However, existing software that is overwritten by newer software can have far-reaching consequences. These changes could affect a variety of applications and libraries alike. This can introduce interoperability-based issues that are not easily diagnosed or fixed. Furthermore, it is also possible that user preferences and system-wide configurations will require re-examination after an upgrade as these too many have changed. Another issue of concern with upgrades is that certain applications may no longer be supported by either the distribution's maintainer or even by the open source community (in which case support and upgrades are no longer available), or both. In either case, when support for an application is dropped, extra work will be required on the part of the system administrator. If both forms of support are gone, then the application will have to be maintained by the organization itself through various support strategies [2].

Upgrades are generally far more wide reaching in terms of potential consequences than updates. Upgrades generally infer that the system's applications, libraries, kernel, and other important subsystems are to be replaced by newer equivalents that may or may not be radically different from their predecessors. Updates on the other hand tend to cause minimal system changes that often represent little to no discernable impact for both the users (and their applications) and system administrator.

Upgrades, in the sense used throughout this text are indicative of an installation program developed by the distribution's maintainer that are intentionally designed and used for upgrading the operating system. Furthermore, upgrades can require a great deal of time to reaccredit and recertify since upgrades generally impact the entire operating system and not just portions of it, as is the case for most updates. Finally, upgrades are performed for a variety of reasons; sometimes to achieve the next level of functionality or sometimes simply because updates are no longer available for a specific version of an operating system.

2.4.3 Updates

Updates are not generally as far reaching as an upgrade as the intention of an update is to provide bug fixes, enhanced features, security, and functionality. Sometimes updates may also provide enhanced performance and system stability. Nevertheless, the changes imparted onto the system should not generally affect other programs, including outdated programs that require recently updated libraries for their dependencies[3]. This is because updated libraries are not generally different enough from their predecessors to cause serious compatibility-based issues.

Most if not all updates are transparent to both the user and the system administrator in terms of the overall system impact. It is rare for system wide and user-based preferences to be changed. Furthermore, updates generally increase functionality, not take it away. However, the possibility is always there.

Interestingly, updates can perceptually require every bit as much time as an upgrade to rollout in order for the operating system to be reaccredited and recertified as suitable for redeployment. This is particularly true when many system components are updated on the system. However, with updates, as in contrast to upgrades, only portions of the system can be updated at any time such as the kernel and its related subcomponents or an application suite.

The key difference between updates and upgrades is that updates generally cease to be able available after a set period. It is common after several years for updates to no longer be available, as the distribution's maintainer will inform customers that support is no longer available and persuade them to upgrade. Updates, as examined in this text relate to packages provided by the maintainer for use with an application specifically for updating. Although manual updating is possible, this is generally considered as manual maintenance, is very time consuming, and not suggested except when required by circumstances.

Updates are generally used to keep an operating system up to date on an intermittent basis before an upgrade is actually available. Sometimes, it may not be desirable to proceed to an upgrade just yet. In this event, updates can be used to not only keep the system up to date but also to potentially exploit some of the newer benefits imparted onto to the newer operating system by the maintainer; however, this may vary as not all maintainers will provide new functionality in their updates.

2.4.4 Manual maintenance

Performing manual maintenance (via manual patching, recompilation, reinstallation, etc.) is far from an impossible task. It is however a time-consuming task, and is sometimes daunting, particularly if the system administrator does not have enough experience. Manual maintenance requires finding a more recent version of an application, tool, utility, or kernel, and through access to source code is recompiled and reinstalled. Furthermore, the latest source code patch can be applied to the original source code and the program can then be recompiled and reinstalled.

The fact that the Linux operating system is modular makes it possible to install newer applications and libraries possible without generally compromising system integrity and stability. This in turn enables a more rapid reaccredidation and recertification so that the operating system can be quickly redeployed. In addition, by maintaining vigorous system version control and actively documenting all changes made to the operating system through manual maintenance, it

[3] However, this will vary from case to case; this is only a general rule of thumb.

will be easier to successfully upgrade various operating system components without affecting overall stability and further help to increase the efficiency of this process.

2.4.5 Issues

2.4.5.1 Potential problems

Regardless of whether an update, upgrade, or manual maintenance is performed, there is always the risk of incompatibilities and/or inconsistencies in the system due to abrupt changes in system libraries. For example, older applications may rely on a set of functions and API's from a given library. However, if that application is no longer supported through a given upgrade and that required library is in some way substantially different from its predecessor the application will very likely become inoperable.

Of course, there are ways around this, but this can only be determined by attempting the upgrade and testing the various applications. If it turns out that the newer library does not provide the required functionality, then the appropriate version can be reloaded onto the system from backup media. However, it should be restored back onto the system in a different location. At this time, the system administrator can create specialized environments using environment variables so that the affected application knows where to correctly locate its required library(ies).

Although this is only an example, it represents the main problem encountered when upgrading (and sometimes after updates) an operating system. Thus, the only way to be sure of system integrity after an upgrade is to test required and other important system applications.

2.4.5.2 Solutions

It is very likely that the issue of system library inconsistencies can best be managed over a very long-term period similar to the Navy's requirements of 15 to 25 years is to perform manual system maintenance. Doing this for an operating system is certainly is far more complex and time consuming than for updates and upgrades.

The main problem with upgrades, as already stated is that there will come a point in which older programs will no longer work because: 1) they are no longer supported; 2) libraries that they depend will no longer be supported, or 3) those libraries will have been changed to newer versions that will not support the older API's and functions. This is never a certainty but it is likely, and the larger the distribution the more likely it will become. This is because larger distributions inherently have more interdependencies due to the increased number of applications.

The main issue with updates is that there comes a point in which the maintainer of the distribution will no longer provide those updates for a version of the operating system. As a given operating system becomes older and is replaced by newer versions it becomes more likely, looking to the future, that at a certain point both support, updates and upgrades, and patches will no longer be released for older operating system versions. At this time, it is highly likely that the maintainer will attempt to convince the customer to migrate to a newer and supported version of the operating system, perhaps by offering financial incentives. However, in upgrading the system, a different set of problems will occur over time, as previously explained.

The problem in maintaining the same operating system for such a lengthy period is that there will inevitably come a time when software must move forward with the times. In most cases, the majority of C2 applications are likely to be custom-built; therefore, it will be the kernel and certain other key open source applications that will have to be maintained over the long-term.

Thus, in order to maintain vigorous control over what is to be changed, replaced, and how and where the binaries are to be installed can only be accomplished through manual system maintenance. While much of the work in manual system maintenance will require the manual reconfiguration, recompilation, and reinstallation of programs and libraries, the system administrator has full control over how the reinstallation is to occur and thus gains much larger control over how the overall system will be affected. Unfortunately, this requires a system administrator with both ample system administration and programming experience in order to be able to adapt to all the various tasks he is likely to encounter. Of course, the system administrator can continue to take full advantage of updates and upgrades in so long as they do not compromise system integrity. *Furthermore, it is highly suggested that the system administrator take full advantage of updates and upgrades as long as they are available and do not compromise overall system stability and integrity; manual maintenance should be left as a final option for maintaining long-term operating system employability.*

The ultimate benefit of performing manual system maintenance is manifest. Through vigorous system, application and binary control it is possible to minimize or disregard any potential changes to the system. By documenting all system actions and changes, it is possible to realistically trace and narrow down the root cause of problems to previously taken actions or changes. Because of this, it is reasonably expected that reaccredidation and recertification of the operating system will be far less time consuming than with updates and upgrades such that the C2 operating system is ready for rapid redeployment.

Furthermore, the emphasis of a trial laboratory in which to perform tests and exercises prior to final deployment aboard the frigates cannot be overemphasized. In this lab setting, it becomes possible to determine most potential issues of contention before they become apparent in a theatre of operation.

2.4.5.3 Tools

Most Linux operating systems come with several pre-bundled tools that allow system administrators and programmers to more closely examine the system calls and interdependencies required by various applications. The main tools are *Ltrace* and *Strace*. Each of these tools performs a different task, but when combined together, they are able to provide a great deal of information about an application and its many dependencies.

Strace can be used to display all the kernel related system calls that an application makes. The system's current system calls be found in various files across the system. Under Fedora Core 6, the kernel's available system calls are found in */usr/include/bits/syscall.h*. Using this command, it is possible to debug various issues that can result from a changed kernel [10, 11]. This command also makes it possible to determine various race conditions that may result from system calls as well as the various libraries that are accessed. *Strace* cannot be used against actual library files.

Ltrace is another type of program [12]. It is similar to *Strace* but it differs in that it is designed to list all the library functions used by an application. It can also be used to list all an application's system calls; however, this option is best left to *Strace*. However, *Ltrace* does not actually provide the name of the library the function requires nor can it be used against library files.

If a program is running in memory, left in the background, or runs long enough to run another command from the same or different command line, more information about which specific libraries are used can be found from */proc/?/maps* and */proc/?/smaps* where "?" denotes the process id (PID) of the application in question [13, 14]. Both files list the basic information although they present it differently. Furthermore, although these files provide no details about the specific functions used by their host application, at least they are very clear about which libraries are actually required. This can be very helpful in tracking down missing or changed libraries.

Ldd [15, 16] is another tool that can be used to list all the libraries a specific application uses. It is very simple to use and does not require parsing through many pages of information as is the case with *Ltrace* [12]. This program is also able to determine library file dependencies making it very useful for determining other possible library-based contentions. The program can be run against both executables and libraries.

Objdump is another very useful program [17] that is used to determine library functions and the originating library. However, the library name is not always stated as the actual file name of the library. Often times, libraries are given proper names and this is recognized and used by *Objdump*. Therefore, combining this tool with *Ldd* [15, 16] will help to sort out any confusion. The program can be run against both executables and libraries.

Lesser-used programs that can be of use include *Readelf* [17] and *Lsof* [18]. It is certain that the abovementioned tools may not be suitable under all circumstances; however, without any debugging information attached to a library or executable, they are an excellent substitute. Furthermore, they are easy to use, even for the novice. More advanced tools are available commercially and are sometimes bundled with specialty development environments for Linux. By combining the information from the various tools, it becomes possible to narrow down many potential problems, and once known, fixing the problem is a more straightforward task.

2.5 Summary

Understanding how modern complex operating systems function can be a cumbersome task. It is not necessary to fully understand all the finer details; however, a basic understanding of dependencies and their affect on the system's overall stability and integrity is important. Many tools can be used alone or combined to provide a clearer of view of dependencies and their interaction with one another. The type of system maintenance that will be chosen to keep the system's kernel (at a minimum) up to date will depend in part on the expertise of the system administrator and the availability of updates and upgrades. There is no clear-cut solution. In certain circumstances, it is more appropriate to apply an update or upgrade. In other cases, these should be altogether overlooked when manual system maintenance is the simplest and clearest solution for a specific update of the kernel, application, or library.

Furthermore, although the Navy has expressed its desire to resist performing operating system updates and/or upgrades, it is very likely that at one point they will no longer have any choice. It is unrealistic for the Navy to expect that manual system maintenance is the solution for all their maintenance needs. The cost in resources and capital will too high and complex, especially at the beginning of a deployment. The Navy has many options available to it for providing various forms of maintenance [2]. In addition, whether only the first few years (most likely for the first half of the C2 system's lifespan), regardless is maintenance is provided through updates and/or upgrades, both are customizable. Upgrades, however, are generally less customizable than updates. Therefore, for at least the first 10 to 15 years the Navy should not have to worry about having to perform manual system maintenance. Unfortunately, once a point has been reached where the vendor no longer exists, no longer provides support, changed its business line or the Navy can no longer approve of newer operating system versions due to excessive compatibility-based issues (or other issues) then the Navy will have little choice but to pursue manual system maintenance. Until that time, however, it is not suggested.

In conclusion, the Navy will very likely have no choice about performing updates and upgrades if they plan to periodically change hardware. Even a system's hardware that is never changed (which over a 15 to 25 lifespan is very unrealistic) must still be periodically maintained for a variety of reasons, as already put forward earlier in the section. A non-maintained system will eventually suffer from software degradation, a condition that all software experiences through long-term use and often the only remedy is software maintenance.

3. Methodology I - backup and restoration

3.1 Objective

The objective of this section is to examine in detail the various issues that must be understood before attempting to undertake the task of preparing and creating full system backups. Backups are vital to restoring the system to an operational state should any system changes fail. Failure can occur during laboratory tests, migrations, reconfiguration, updates, upgrades, or manual system maintenance. The importance of having a laboratory for testing purposes cannot be overemphasized. In this test environment various backup and restoration procedures can be examined to ensure the applicability and usefulness of a given backup-restoration scheme, as well as ensure data integrity before proceeding with Part II.

3.2 Backup considerations

3.2.1 Plan development

A plan should be developed prior to performing any backup of the computer systems and their filesystems. This plan should take into account the various facets necessary for a successful backup as are examined in the following sections. A backup plan will be the result of a methodological analysis that is a synthesis of the organization's operational policy and requirements that also includes the computers' operating system capabilities and requirements. In addition, a backup plan must be both flexible and adaptable, as contextual changes that are likely to occur must accommodate for unforeseen errors and mistakes. These contentious problems can be introduced by hardware, software, and human error; a robust plan will provide for alternate ways around them in order to successfully perform a backup. The backup plan will have a direct impact on the restoration plan (Section 3.4.2) that will itself have to overcome many of the same challenges, thus ensuring the continual availability of data. Finally, as with any plan, it should not be put into action until it has been documented, analysed, reviewed, and tested.

3.2.2 Tools

The tools *Dump* and *DD* are generally best suited for performing operating system-based backups. However, other tools such as *Tar* and *Cpio* are more suitable for user and application-based backups. However, different contexts will require different tools for the specific task. The capabilities and ineffectualness of the various tools are examined in the ensuing sections. Although *Dump* and *DD* are preferable for operating system-based backups, *Tar* and *Cpio* have their own specific uses and capabilities. It is assumed that the reader is familiar with the aforementioned tools and understands how to implement them. Finally, no commercial backup tool has been examined in this Technical Note in order to maintain tool uniformity with respect to the various Linux distributions currently available on the market.

3.2.3 Data-related factors

An important factor for determining the appropriate backup scheme is the type of data to be backed up. The data type, its location, availability, and correlation to the operating system will largely determine which tool or tools can be used. The following subsections are an in-depth examination of these various factors.

3.2.3.1 Data type

A determining factor for selecting the appropriate type of backup scheme will depend on whether the data is operating system, user, or application-based. Furthermore, the data type will influence the selection of the backup tool. Operating system data should only be backed up using *Dump* or *DD* (for reasons to be examined further on) while user and application-based data can generally be backed up using any of the four aforementioned tools. These issues are examined in the different subsections below.

3.2.3.2 Special attributes

A determining factor for selecting the appropriate type of backup scheme will depend on whether there are special filesystem attributes that need to be backed up. Backup tools *Tar* and *Cpio* cannot capture special file and filesystem attributes. These files can, however, be backed up using *DD* and *Dump*.

3.2.3.3 Devices

A determining factor for selecting the appropriate type of backup scheme will depend on whether there are device files to be backed up. Devices files can be backed using all of the various tools. However, if the devices are locked, in use, or are automatically skipped by some of the tools then *DD* and *Dump* can be used. Since device files rarely use extended attributes, assuming the device files are accessible, the tools *Tar* and *Cpio* should be able to backup them up.

3.2.3.4 Raw data

A determining factor for selecting the appropriate type of backup scheme will depend on whether any raw partitions or data need to be backed up (i.e. raw database partitions). *DD* is uniquely suited to this task. *Dump*, *Tar* and Cpio cannot be used because there is no recognizable filesystem for data acquisition from these partitions or devices.

3.2.3.5 Locked files

A determining factor for selecting the appropriate type of backup scheme will depend on whether locked files need to be backed up. Certain key operating system files from */dev* and */proc* are automatically locked by the kernel and normally not accessible by most of the backup tools. These files are also generally off limits to users and applications (and sometimes the root user). Consider the following:

a. Are there locked operating system files? *DD* is generally the only way to acquire them using a partition dump if the *Dump* program cannot acquire them, but this should only be done if the filesystem is inactive otherwise image acquisition may result in a corrupt image. It is preferable to run *DD* only when the partition/disk is offline or mounted read-only.

b. Are there user or application-based locked files? Can those files be backed up without resulting in file or data corruption? Generally, other than for database files, locked application and user files can be backed up using *Tar*, *Cpio*, and *Dump*.; however, locked operating system files cannot.

c. If some files must be backed up and are locked and inaccessible no matter what is attempted then a rescue or Live CD can be used to backup the files.

3.2.3.6 Data file volatility

A determining factor for selecting the appropriate type of backup scheme will depend on whether the files to be backed up contain volatile state-based information about the operating system or some arbitrary application or service. Consider the following:

a. Operating system files that contain volatile data from directories such as */dev* and */proc* should only be backed up using *Dump*. Many of these files, especially from */proc* contain very volatile system and application data (i.e. RAM, system and application states, etc.) and must be treated carefully.

b. Some of the files are application and service-based and can be safely backed up when the offending application or service is disabled. Nevertheless, *Dump* can normally handle these files, even if they are left running.

c. Device files (*/dev*), however, sometimes can be problematic. For device files that must be backed up but that cannot be accessed, disabling a specific device will often work. Dump can normally backup these files even if they are in use. However, in case where they cannot be backed up *Dump*, *DD* can be used, but only if the filesystem is quiescent.

d. If for some reason certain files that must be backed up cannot be accessed regardless of what is attempted, then the system must be placed offline and backed up using a rescue or Live CD.

3.2.3.7 Running applications and services

A determining factor for selecting the appropriate type of backup scheme will depend on whether application-based files can be backed up while the application is left running. For example, it is generally not advisable to backup a database file while its corresponding application is running; buffered data may not have yet been written out to the database, potentially resulting in data loss upon restoration. Consider the following:

a. An application of service left running can normally be backed up as is. *Tar* and *Cpio* are well suited for application-based backups. *Dump* can also be used although it is more of an all-purpose tool.

b. If the application is a database application, then the application and its files can normally be backed up. However, database files should never be backed up while their corresponding applications are running.

c. Many small database files can be backed up while their corresponding applications are running. This however, will however, depend on the type and size of the database. Larger databases and should always be shut down prior to backing up. This is due unwritten buffered data in memory. For smaller databases, unwritten buffered data may result in nothing more than several minutes' worth of data loss. Larger databases, however, may be inconsistent and be rendered unusable if buffered data is left unsynchronized.

d. Is the application data stored on partitioned or raw devices? *Tar*, *Cpio*, and *Dump* will on partitioned devices where a valid filesystem structure exists. *DD* is best suited to raw devices.

e. Are the applications services? They too can normally be backed up. However, services often write data out to log files and these log files may not have been recently synchronized such that when backed up may not contain unwritten buffered data, resulting in partial data loss. If a service's log file(s) is important then the corresponding service should be shutdown prior to backing it up. Normally, however, backing up services and their various files is not an issue.

f. Are user data files currently in use by applications? Normally, user data files (i.e. word processing, spreadsheets, etc.) that are in use can be backed up. However, much will depend on file sizes and how often buffered data is synchronized. Normally, this will not result in corrupted files but may result in partial data loss. Therefore, in use user files can be backed up.

g. If certain files are locked and the corresponding application or service cannot be shutdown, then certain files may have to be skipped. If the filesystem is active then *DD* can be used to acquire an image or the system can be taken offline and backed up from a rescue or Live CD.

h. Are files being shared over the network? These files can also be backed up. However, if they are in are being edited or modified then what is in memory may not be consistent with what was backed up unless unwritten buffered data are synchronized out to disk before backing them up.

In summary, it should be considered a good practice to shut down all non-essential user-applications and databases before backing them up. If they must be left running, ensure wherever possible that unwritten buffered writes are synchronized to disk.

3.2.3.8 Active operating system

A determining factor for selecting the appropriate type of backup scheme will depend on whether the operating system can be left running while backing it up. This will depend on a host of issues such as file types, special file locking mechanisms, and the availability of devices and files. Generally, most if not all operating system files can be backed up while the system is running; however, it may be necessary to shut down certain services and user-based applications before proceeding with a backup. Consider the following:

a. What is the availability of the operating system's files? Are they all available? Are some unavailable? Can the backup proceed without those files? Generally, once applications and services have been disabled (only if necessary) the only inaccessible files are volatile and device-locked files that are not generally required for operating system backups. However, if one or more of these files must be backed up, then an offline system backup using *Dump* or *DD* can be using a rescue or Live CD.

b. If necessary, can the operating system be shutdown for a given period to back it up? This may depend on organizational policies and operational requirements. An inactive operating system and its files can always be backed up when it is offline, regardless of file type, data contents, access lists, permissions, etc.

c. Can portions (i.e. partitions) of the operating system be taken offline or made read-only in order to decrease backup and system downtime while other online parts are backed up? For example, */usr* or */var* could be taken offline since few to no operational changes are made here while the system is running.

d. Is it important to backup the more volatile portions of the operating system such as */proc*? Generally, the answer is no as most of its contents are volatile and dynamic and is only required by the running kernel. Only certain static text configurations are worth backing up from here, but these configurations should always be made from start-up and initialization files and scripts. However, if necessary *Dump* can often be used for backing up the contents of */proc* without taking the system offline.

e. If certain files cannot be backed up while the system is online will a restoration be successful without those files? If not, are there are other available sources for those files? If not, then the operating system will have to be taken offline in order to backup those files.

f. Is the operating system spread across multiple disks and/or partitions? If so, then often these operating systems are easier to backup than single-partition systems. There are many various reasons why this type of system is easier to backup. *Dump* can be used for almost all the partitions and where it cannot be used, if a given partition can be brought offline or is inactive, and then *DD* can be used.

g. Many other factors exist that must be contended with that go beyond the scope of this subsection. Whenever system volatility and file accessibility-based issues are present, backup procedures will generally be more cumbersome than they would otherwise have been. Thus, there is unfortunately no ubiquitous solution for operating system backups, only practical solutions.

3.2.3.9 Filesystem availability

A determining factor for selecting the appropriate type of backup scheme will depend on whether filesystem(s) can be placed offline. This may vary according to various system requirements and organizational policy. Consider the following:

a. Can the files be safely backed up if the filesystems are mounted? If so, then *Tar*, *Cpio*, and *Dump* can be used. *DD* should never be used on actively mounted partitions unless the filesystem(s) is quiescent.

b. If one or more partitions are placed in read-only mode, can the operating system carry on for the duration of the backup without affecting system stability? If so, then *Tar*, *Cpio*, *Dump*, or *DD* can be used.

c. Will placing filesystems in read-only mode affect applications, services, and user data files? If so, then either backups should be performed during off-hours or if this is not possible, disable the affected services and applications.

d. If one or more partitions are placed offline can the operating system carry on for the duration of the backup without affecting system stability? If so, then *Dump* or *DD* can be used.

e. Will placing filesystems offline affect applications, services, and user data files? If so, then either backups should be performed during off-hours or if this is not possible, disable the affected services and applications.

f. Are locked files preventing filesystems from going offline or being made read-only? If so, then by disabling the service or application using the locked file(s) may correct the problem. If it does not, then the system itself may have to be placed offline in order to obtain a backup of the affected filesystem.

3.2.4 Other backup factors

3.2.4.1 Media

A determining factor for selecting the appropriate type of backup scheme will depend on the type of backup media that will be used, its location, its path, and its availability. The backup media can be local or remote to the system to be backed up and it can be a disk or tape. It is important that media access always be available under varying conditions. For example, if data is normally backed up to a local tape drive and it fails, then a remote tape drive can be used but only if the network path is available and permissions are set for sending and receiving a remote data stream. Consider the following:

a. Is enough media available, especially if tape is used? Is the backup tool multivolume capable? *DD* is not multivolume capable but it can be made so through scripts.

b. Can the media be read and written to by other devices? For example, is the media universal enough that other devices (i.e. tape drive) can read and/or write to/from it?

c. Are spare local and remote backup media and devices available to parallelize the backup if multiple disks and/or partitions can be simultaneously archived? The backup tool does not need to be multivolume capable in this case, only multi-device capable (achieved through scripts).

d. If the media and/or archival device are remotely located, are the necessary network permissions appropriately granted for data streaming and reception?

e. Are the device, media, and network paths available? Are there specific schedules for their usage or are they available 24x7? Can they be rededicated for other tasks?

f. Is the media checked for errors prior to usage? If not, can this be automated using scripts? Does the backup device perform error or CRC-checking?

g. Are different types of backups stored on different media? For example, are backups done at different times placed on the same or different media? If an incremental and differential backup of one or more systems is done, are they too placed on the same media?

h. Are different types of data stored on different media? For example, will operating system backups be placed on the same media as user data backups?

3.2.4.2 Lifespan and storage

It is important to determine how long data should be kept. Depending on the data and its function, it may be necessary to keep it for several weeks, months, or longer. For example, legal documents by law must be kept for a prescribed period. Consider the following:

a. Is the data to be kept for a short or long period? Are there any laws that mandate how long certain types of documents must be kept? What is the organizational policy concerning backup and data longevity? Is the policy the same for user data and operating system data?

b. How long must different types of data or backups be stored? For example, is user data stored alongside operating system data? Are they stored on the same media?

c. If the data is stored for a long period there are various storage issues and requirements to examine. For example, will the media be stored in a temperature and humidity controlled environment? Is it secure, safe from fires, floods, and tampering? Who has access to it? Will it be stored onsite or offsite?

d. Will data be stored incrementally or differentially? This may depend on required data longevity and organizational policy. Is there a preference for incremental or differential backups? All the tools but *DD* can be used for incremental or differential backups. How often is a full system backup (operating system and user data) carried out with respect to incremental and/or differential backups? These answers will help to determine how much media is required.

e. Are the users involved in helping determine the lifespan and usefulness of their data? Is there a policy in place for maintaining user data backups for set periods? For example, should users be consulted on an intermittent basis to determine if their data and storage requirements have changed?

f. What is done with users and their data when they are no longer system users? Are the accounts destroyed and all data saved to CD or DVD or must the accounts be disabled and data stored and backed up with all the rest of the user data?

g. How often are full backups performed? Are they incremental or differential? Once a new full backup is performed, existing incremental or differential backups could be overwritten and reused again for newer incremental or differential backups.

3.2.4.3 Data security requirements

Before carrying out any backup, it is important to determine security requirements that are necessary for safeguarding data, both during backup and storage. Consider the following:

a. Will data be sent over the network? If so, is it on a trusted network? If not, then perhaps data should be encrypted; this can be done by piping backup the data stream through *SSH* or *GPG* before it goes out onto the network.

b. Should the data be encrypted, regardless if it is backed up locally or remotely? If so, then the data should be encrypted from the source system using *GPG* or other similar tool.

c. If encryption is used, is there a mechanism or resource to help in the management of encryption keys (i.e. PKI)? Who has access to the keys? Who has the passwords necessary for encryption and decryption?

d. What network tool should be used for network backups? *RSH*, *SSH*, and *Netcat* can be used. *SSH* should be used for encrypting the network stream between computer systems. *RSH* and *Netcat* can be used on trusted networks or if the data stream is already encrypted via other means (i.e. *GPG*).

e. What are the current security settings for the machines involved in the backup process (local sources and remote systems)? What are security settings supposed to be set to? Can unauthorized users or systems gain access to either the source and/or remote systems while backups are being performed?

f. Can the backup tool support output/input piping? This may be required, depending on the type of backup to be done, including compression and encryption requirements as well as data backup location (local or remote backup system). *Tar*, *Cpio*, *Dump*, and *DD* all support piping. Advanced piping capability can be achieved through scripts.

g. Does the backup user have the necessary rights and permissions to backup all system and user data? Does that user have permissions to send and receive backup data streams over the network from/to remote systems?

3.2.4.4 Size requirements

It is important to determine the various size requirements of the data before performing any backup. Consider the following:

a. Is there enough backup media available for the backup to complete correctly?

b. Is the backup going to require multiple volumes? If the backup cannot be stored onto a single backup media will the tool require multivolume capability? *DD* can be made multivolume-capable using scripts.

c. Is backup space a concern? Does the backup device support hardware compression? If space is a concern and the backup device does not support hardware compression then through piping any of the backup tools can be made to send the backup stream through *Gzip* or *Bzip2* for compression before redirecting the data stream to the backup device.

d. Large datasets such as operating systems or databases should be backed up using *Dump* or *DD* because they are much faster than the other tools. *DD* is efficient if the device (partition) is more full than empty because it copies all bits from that specified device.

e. Small datasets and user data can be easily backed up using *Cpio*, *Tar*, or *Dump*. If there are no constraints such as locked files, special file attributes, etc., then *Cpio* and *Tar* may be more appropriate then *Dump*. However, larger datasets should use *Dump* wherever possible.

3.2.4.5 Data accuracy and relevancy

It is important to determine how accurate the data to be backed up must be prior to performing any backup. Consider the following:

a. Is the exactness of the data important? Must if be exactly as it appeared on disk? If so then *DD* should be used to acquire a bit-copy image.

b. Are raw partitions to be backed up? If so, raw partitions must be bit-copied to ensure accuracy as there is no recognizable filesystem on them; *DD* should be used for such a situation.

c. Are the filesystems too large to be backed up to one media? If so, then the backup should be done using a multivolume capable tool or script. Furthermore, the multivolume backup should contain all the data from that specific filesystem.

d. Does the backup device support CRC or error checking when writing data? When reading data? Can the backup tool perform CRC or error checking? Can it deal with errors and if so, how? How much data will be lost if bad blocks are found on the media.

e. Does the backed up data contain all the same filesystem attributes as the original data? Does the tool understand how to treat extended filesystem attributes?

f. Can the backup tool accept user-provided input for determining which files to backup?

g. If disk exactness is not necessary then the tools *Dump*, *Tar*, or *Cpio* can be used.

3.2.4.6 Speed and bandwidth

It is important to determine the network speed, amount of time available and required by the tool, and required network bandwidth. Time constraints will also be a determining factor in determining which backup utility to use. Consider the following:

a. *Tar* and *Cpio* are very fast when backing up small to medium-sized datasets. *Dump* is very fast for medium to large-sized datasets. *DD* is not fast because it must copy every bit of data from a partition or raw device.

b. How much network bandwidth is available? Does it decrease during the day and increase at night or on weekends? If so, then backups should be scheduled when network bandwidth is at highest.

c. If backups must be performed during periods of high network utilization, then data compression tools can be used to help reduce the amount of required network bandwidth. The data stream can be compressed using *SSH*, *Gzip*, *Zip*, or *Bzip2* before being sent and decompressed at the remote system.

d. If *DD* must be used then it will typically require more network transmission time as bit-copies are larger than their counterpart backup types. Thus, *DD*-based backups should be compressed and backed up during periods of low network utilization.

e. What is the read/write speed of the archival media and/or device? This could be the backup process' bottleneck. Spreading the backup from multiple partitions/filesystems across multiple backup devices (local or remote) could significantly speed up the backup. However, this is not a useful option for multivolume backups; instead, it is better suited for simultaneously backing up multiple datasets.

f. Local backup devices are usually faster than remote devices and using multiple backup devices simultaneously for large datasets from different partitions is faster than using a single backup device, whether local or remote.

g. SCSI devices are generally faster than IDE devices and are therefore highly suggested.

3.2.4.7 Tool summary

It is important to determine the capabilities of the various tools examined within this section. The following provides a brief summary for these tools:

a. Will data be sent over the network for backing up to a remote system? If so, then *RSH*, *SSH*, and *Netcat* can be used for sending and receiving network data.

b. Is the backup to be multivolume? If so, then *Tar*, *Cpio*, and *Dump* can be used. *DD* can only be made multivolume capable through scripts.

c. Is data encryption required? If so, then *GPG* can be used.

d. Is network encryption required? If so, then *GPG* or *SSH* can be used.

e. Is data compression required? If so, then *Gzip*, *Zip*, or *Bzip2* can be used.

f. Is network compression required? If so, then *SSH*, *Gzip*, *Zip*, or *Bzip2* can be used.

g. Is piping required? If so, then *Tar*, *Cpio*, *Dump*, and *DD* can be piped into any and all of these programs: *RSH*, *Zip*, *SSH*, *Netcat*, *GPG*, *Gzip*, and *Bzip2*.

h. Is error checking required? If so, then *Tar*, *Cpio*, and *Dump* support some form of error checking. *DD* can perform error checking using scripts.

i. Is the operating system to be backed up? If so, *Dump* or *DD* should be used.

j. Are user and application data to backed up? If so, then *Tar*, *Cpio*, or *Dump* can be used.

k. Is the dataset large? Then *Dump* or *DD* should be used.

l. Is the dataset small? Then *Tar* or *Cpio* should be used.

m. Are incremental or differential backup required? If so, then *Tar*, *Cpio*, or *Dump* can be used.

n. Is error checking is required? *Tar*, *Cpio*, or *Dump* perform some form of error checking.

o. Are there locked files? Are there volatile files? If so, then *Dump* should be able to handle them.

p. Are applications and user data in use? *Tar*, *Cpio*, or *Dump* should be able to handle this.

3.2.4.8 Resource availability

It is important to determine the availability of various backup resources prior to performing any backup-related action. This will help to better plan and schedule resources. Consider the following:

a. Are centralized backup servers available? If so, a backup server can be used to centralize and facilitate backups from one or more systems. However, the more systems that stream data to a centralized system at the same time will cause a network bandwidth slowdown.

b. Does the centralized system have more than one backup device? If so, then more than one system can backup its data at the same time. However, performing multiple backups and/or restorations at the same time may be slower and more tedious to carry out.

c. Is the backup to occur over the network onto a remote system? If so, can the network support the backup system's required network bandwidth? If not, then multiple network adapters with different network connections will be needed to handle the network load.

d. Are spare backup systems and devices always available? This is important if a backup fails due to hardware. Having extra remote systems (if they are used) and extra backup devices will facilitate and speed up the backup process should a hardware failure occur.

e. Is there enough spare media? Every so often media is found to be defective, even new media.

f. Is the amount of data to be backed up very large? If so, then it may be more appropriate to backup the system's various filesystems simultaneously onto multiple backup devices (local and/or remote devices) to speed up the process.

g. Are backup operators available to intervene if backups should fail or require a specific action to be taken? This is unlikely to be found in small organizations; however, large organizations often have backup operators 24x7.

h. Are the archival devices capable of handling multivolume backups? Robotic tape libraries can handle multivolume backups. If the device is not a tape library then a multivolume backup can be performed using scripts that divide a backup among multiple backup devices.

3.2.4.9 Backup schedules

It is important to determine when backups can be scheduled before proceeding with any backup-related action. Consider the following:

a. When are the systems to be backed up? Are they to be done at night, during the day, or on weekends? Backups should be done when all necessary resources are available.

b. Are network resources available 24x7 for remote backups? Is there more bandwidth available on weekends and at night? If so, then backups should be scheduled for these times.

c. Are backups planned to coincide with increased network bandwidth availability? If backups are to be performed over the network and there are large data transfers to occur between one or more systems to a backup server, then backups should be carried out when network activity is low.

d. Do actively used user files and applications affect backups? If so, then backups of user files and applications should occur when the users have logged off. However, if in use files and applications do not affect backups then they can be scheduled for any time.

e. Do certain operating system services and/or applications interfere with the backup process? If so, then a schedule should be made for when they can be disabled so that the operating system can be backed up. If they do not interfere, then they can be backed up at any time.

f. Are certain key files or data locked or unavailable? If the backup can continue without those files then the backup should be made when ever possible. However, if the files are required for a successful backup and the tools *Dump* or *DD* are not able to acquire them then a time should be scheduled to offline the system for backup using a rescue or Live CD.

g. How will backups be scheduled and executed? If they are using the system's scheduler then backups can be automated. If they are executed from the command line then a backup operator will be required. In either instance, scripts can be used for automation.

3.3 Filesystem checking for backups and data restoration

3.3.1 Filesystems

All filesystems, no matter how robust, modern, or technologically savvy are all at risk for data loss and/or corruption. The most likely time for this to occur is during disk data writes that simultaneously occur with a power outage or fluctuation. Although modern filesystems are able to reduce this danger by incorporating filesystem journaling, the risk is always present. In addition, unrecoverable hardware errors and kernel panics may also be a likely culprit. This section therefore examines various filesystem-checking tasks that should be performed prior to undertaking any backup-related action.

3.3.2 Issues

Filesystem checking, although very important for backups are equally important for data restoration. Restoring files from a good backup to a damaged filesystem could result in the overwriting of existing files or even further corrupt the filesystem. Similarly, a corrupt backup due to filesystem damage will likely result in missing files upon restoration, assuming that the archive is not corrupt because of the damaged filesystem. Some readers may assume that these issues are exaggerated; however, a simple lookup on the web for filesystem errors with respect to backups and restoration will yield much useful information.

3.3.3 Periodic checks

Filesystems must be checked periodically; even the simplest filesystem inconsistency can cause file damage. It is also possible that severely damaged filesystems may be missing files or they may become inaccessible due to damaged permissions and/or ACL's. The only way to verify the status of a filesystem and its files is to perform a filesystem check. Linux provides the necessary tools for all of its supported filesystem formats, the most notable of which is *Fsck*. However, certain filesystems are more sensitive than others are. For example, Ext2 is far more sensitive to power outages and incomplete data writes as compared to its modern enhancement, the Ext3 filesystem. Although similar, Ext3 employs a journaling mechanism that ensures safe data writes resulting in a more consistent filesystem.

In addition, certain backup tools are more sensitive to filesystem errors than others are. *Dump* and *Restore* are particularly sensitive as they require (*Dump*) and recreate (*Restore*) filesystem data structures. These data structures are stored using filesystem i-nodes and damage to them can result in damaged or inconsistent backups. It is also possible for backups to crash and result in incomplete archives due to filesystem errors; this can occur for all the backup tools (except *DD*). However, *Tar* and *Cpio* are generally not as severely affected by errors as *Dump* and *Restore* are. Nevertheless, all these tools can suffer from filesystem inconsistencies. *DD*, however, is not susceptible to filesystem errors as it will simply copy over all filesystem errors to its image file that can when mounted (*DD* images can be mounted) be checked.

3.3.4 Repairs

The Linux operating system supports and provides all of the necessary tools for checking and repairing a variety of filesystem formats and all but the most troublesome errors can be found and fixed. It provides the *Fsck* utility for performing filesystem reparation. Although filesystem checking is important, there are certain dos and don’ts. The most important don’t is that a filesystem should never be repaired when it is mounted, even if it is mounted in read-only mode. Even when a read-only filesystem is mounted, buffered changes made to it are likely to result in a more serious problem than that which was fixed if the disk is remounted as read/write. Filesystems can be checked (but not repaired) when they are mounted; however, this can be dangerous, depending on the filesystem type and the level of disk or partition activity and underlying damage. It is always safest to check a filesystem when it is mounted read-only and it is always safest to repair a filesystem when it is offline. *DD* image files can also be checked and repaired just as with any other valid partition or filesystem and the same basic rules about checking and repairing apply. Severely damaged filesystems (if a backup is not available) may require forensic tool recovery to reconstruct damaged data and/or filesystem structures. Finally, the root filesystem should always be checked and repaired in single-user mode (during system start-up) or from a rescue or Live CD. Under no circumstances should the root filesystem be repaired while the operating system is active and operational.

3.3.5 Scheduling

It is important that time be made periodically for checking a system’s filesystem(s); often the best time to do this is at system start-up. However, some systems, particularly mission critical C2 systems may have operational requirements that make taking them offline difficult. Nevertheless, periodic maintenance is required even on these systems. If time cannot be periodically scheduled for certain systems then filesystem checks and repairs should be made before performing other system maintenance tasks such as updating, upgrading, and/or replacing/adding hardware. Although certain filesystem formats are almost “impossible” to corrupt (i.e. XFS) due to the many inherent safeguards they incorporate they should still be checked as there is no such thing as a perfectly stable filesystem since they are always changing and in a constant state of flux. In addition, if it is not already done, it is important to incorporate filesystem checking into organizational policy regarding system maintenance. Finally, it is better to have certain systems or services unavailable for brief periods rather than have an inconsistent or corrupt backup or restoration when it really matters.

3.3.6 Filesystem formats

Linux supports multiple filesystems formats and provides the necessary tools for fixing them. However, how well a given tool supports, checks, and fixes one or more filesystem errors will depend largely on the maturity of the tool and its underlying filesystem. All of the major filesystems commonly used by Linux are fully supported. In addition, Linux supports the mounting of loopback virtual filesystems (i.e. *DD* image files, ISO files, etc.).

3.3.7 Bad blocks

Although much attention is paid to the filesystem, it is also important to check a disk drive's surface for physical errors. Disk drives can suffer from the effects of excessive wear and tear due to the reading and writing of the disk's heads. This wear and tear can result in the formation of bad blocks. These bad blocks are damaged physical disk sectors that contain actual information; however, that information may or may not be useful just as it may or may not reside on unused disk space. What information is contained within a disk block will depend on where it resides with respect to the underlying filesystem. Although a few bad blocks by themselves will not result in an inconsistent filesystem, they can cause files to be damaged, missing, or corrupt. The Linux operating system provides a tool to check for bad blocks, repair physical errors, and move filesystem data and files to good blocks. The program provided by Linux is the *Badblocks* disk-checking program.

It is important to perform bad block checking before any errors become disruptive to the operating system or the users. However, bad block checking can only be done on unmounted filesystems, and in the case of the root filesystem, it can only be done from a rescue or Live CD. This can be further complicated by the operational requirements of C2 computer systems. Nevertheless, time should always be made at periodic intervals to perform bad block checking; a good time will be to perform this task when performing other system maintenance-related tasks. However, the type of disk drive used will dictate when and how often checks should be made. Some advanced SCSI disks offer the ability to dynamically check and repair bad blocks as soon as they are found; however, this feature is only available a select few disk drives.

3.4 Restoration considerations

3.4.1 Plan development

A successful restoration plan should be based largely on the backup plan (developed in Section 3.2.1) that should already have examined the different facets involved in backing up computer systems. These facets include organizational and operational policy and requirements, as well as the operating system capabilities and requirements. How a restoration unfolds is largely dependent on how the backup itself is done; this can include backup devices, the use of centralized backup servers, the type of media and filesystems, skipped files, network paths, etc. Restorations are important to recover from data loss, corruption, or operating system error; however, a system that improperly backed up may cause restoration failure and the files may be permanently lost. Consider that if a particular backup is difficult to perform due to various reasons then it is more likely that the restoration will also be difficult to perform; conversely, a backup done with ease should be also be easily restored. A restoration, as with any backup, can go awry for different reasons, the majority of which can be mitigated or altogether removed if adequately planned. The previous sections, including the following one will help for planning a successful restoration.

3.4.2 Tools

The tools used for data restoration are the same ones used for performing data backups. *Tar* is fully capable of creating and archiving data into a *Tar*-based archive; it can also extract data from the archive. It is the same for *Cpio*-based archives. *Dump*-based archives are restored using the *Restore* tool; it is a command-line and interactive tool used for extracting files from a *Dump* archive. Finally, *DD*-based images can be restored using *DD* by either bit-copying the data back to a raw partition or device, or using the mount command its contents can be made accessible as with any other filesystem. All of these tools are able to read and write from disk and tape-based media.

3.4.3 Various restoration factors

3.4.3.1 Resource allocation and assurance

It is important prior to any restoration that all required resources are made available and allocated (if immediately necessary). For example, resources could be centralized backup servers, additional disk or tape drives, the backup media itself, or personnel such as a backup operator. It is important that resources already be available before the restoration is to commence as unforeseen delays will inevitably lead to increased down time for systems and mission critical systems can only be taken offline for short periods. For example, if a disk fails on a mission critical server while a restoration is underway then a spare disk must be available and someone must be present in order to intervene and to take necessary corrective measures.

Network paths, centralized servers, remote backup devices, etc., should also be available when required, and if necessary, tested beforehand to determine their functional status. In addition, spare hardware should always be on hand in order to mitigate and resolve hardware-based problems as they arise. Finally, it is important to have a rescue and Live CD available so that should the operating system of the system being restored fail then the restoration can nevertheless continue with minimal interruption.

3.4.3.2 Scripts

All scripts used during the backup phase should also be made available for the restoration process. In addition, most scripts, when written for backing up one or more systems using one or more tools will generally require a rewrite to be able to restore data from media using said tools. The restoration scripts, if any, should perform the inverse function of the backup script(s).

3.4.3.3 Data safeguarding

All backup media should be safeguarded and stored in cool environments as stated by the manufacturer's long-term storage recommendations. Generally, this will require controlled environments that have the appropriate mixture of temperature, humidity, and stored away from strong magnetic sources. In addition, the media should be stored according to its value. High-value data should be stored nearby but off-site, possibly in fireproof or fire-resistant storage containers, lockers, or safes. Furthermore, the media should always be secured from theft and tampering. It is also important to consider placing spare equipment such as backup devices and disks in lockup so that when and if they are required it is certain that they will be there.

3.4.3.4 Media testing and device diagnostics

All backup media should be tested periodically to ensure an error free state. In so doing, this will enable the media to be used at any time without worrying about added complications during data restoration. However, exactly how the media should be tested will depend on a variety of factors. For example, the type of media employed used and the employed archive format will be important determining factors. In addition, various tools can be used; *DD* can always be used on tapes to perform tape dumps, regardless of the archive type. The archiving tool, of course, can always be used on its respective archive file. If the archives are stored on disk, then other tools such as *Fsck* and *Badblocks* can be used.

Obviously, the more media there is to test the longer testing will take. Testing time, however, can be reduced by using scripts, and this can be amplified by using automated tape libraries that can be controlled by a simple program (i.e. *Mtx*) via a script. In addition, disk checking can also be automated using scripts. However, an important to question to answer is how often should the media be tested? Another is should the entire media be tested or individual archives? Often, testing once per year will suffice. Testing can often be done by dumping a table of contents of the archives; this is the default most tools, although full data extraction is always a possibility as well. If tapes are used as the main form of media, then they should be re-tensioned periodically to ensure that the tapes do not stick.

It is also important to test the tape drives, disk controllers, and disks (if media is on magnetic disk) using standardized procedures and best practices; often, the devices' manufacturers may provide useful insight and tips. It is important to implement an organizational policy if one is not already in place. Powerful and simple diagnostic tools are available for Linux (*PC Doctor for Linux*) and some standard tools come bundled with it for testing various devices and components.

3.4.3.5 Alternate methods of restoration

It is important to ensure, as with backups, that the restoration plan examines and outlines various alternative methods for restoring data back to the required system(s). As with backups, it can happen that backup devices or network paths become unavailable and another method for backing up the system(s) must be found. It is the same with data restoration; unforeseen and uncontrollable circumstances can cause data restorations to become very complex and cumbersome, and when being performed against mission critical systems, time is essential. For example, if certain disks or filesystems cannot be brought offline on the system marked for restoration, then data can be restored to an alternative directory (on the system or on the network somewhere else and shared) and copied over at a more appropriate time. Alternatively, if network restorations are performed but the network becomes inoperable then it is important that the backup device and media can be easily connected and reconfigured for the system that is to have its data restored.

3.4.3.6 Procedure testing

It is very important to test all restoration procedures before ever requiring them. Testing should be done in a laboratory or other controlled environment where different tests and scenarios can be experimented on without affecting the operational network. Procedures should be tested to determine their suitability and adaptability to changing contextual situations in order to understand how to improve upon them (when and where necessary) so that they can be made to accommodate multiple uses. In addition, by testing them it becomes possible to understand how they will behave due to varying circumstances caused by, for example, a lack of system or human resources, device failure, and network unavailability. However, not all procedures are equally useful nor are they equally applicable in all circumstances. For instance, backed up system data that is stored on tape that is no longer accessible due to a failed local SCSI controller will have to be restored over the network. Another example would be a system's main operating system and boot disk that has failed. It will have to be replaced with a spare disk and its data reloaded from backup; this will require booting from a rescue or Live CD. In the latter example, procedures should already be in place for how to use alternative boot devices and how to restore data using these devices. Regardless of the varying circumstances, it is important that the backup media used can be recovered using the various procedures and these procedures should be found within the restoration plan. The most difficult issue about a restoration is adapting to varying problems and circumstances when time is running out for a mission critical system; a set of procedures will likely help to better adjust to such circumstances.

3.4.3.7 System and filesystem availability

Depending on the type of data to be restored it may be possible to restore it directly back to the filesystem(s) without interfering with the system's ongoing operations. However, this will depend greatly on the type of data to be restored. Generally, applications, services, and user data can be restored without ever having to take the system offline. Certain applications or services may require being taken offline temporarily while their data are restored. In addition, depending on the type and amount of user data to be restored it may be necessary for those users to logoff while their data is restored. It is also entirely possible that all the users may have to logoff, depending on how much user data is to be restored and whether it will affect some, most, or all of them. If the backup contains operating system data, however, then it is possible that the system will have to be brought offline in order to conduct a restoration; this too will depend on whether the backup must overwrite system-locked files or other key operating system files that must not be changed while the system is operational. Thus, a determining factor in whether the system or filesystem is to be taken offline will depend greatly on the type of data to be restored.

If a backup was done using DD then the filesystem must be unmounted prior to restoring its data, or it can be mounted using the system's loopback device feature and mounted as a virtual filesystem and accessed from there. If the data was backed up using Dump then the filesystem can usually remain mounted in so long as the files to be overwritten are not in use. *Tar* and Cpio-based backups can generally be restored without problem in so long as the files they overwrite are also not in use. In certain cases, an alternative to using another boot device for performing a restoration, the system can be brought into single-user mode and restored from there. However, if the data resides over the network then procedures should exist for how to access the network and data from within single-user mode. In certain cases, where specific files must be restored along with the rest of the restoration that cannot be executed because certain files cannot be overwritten when the system is active, a rescue or Live CD will have to be used. Finally, most restorations can be made to filesystems while they are actively mounted; however, this will depend on both the type of backup and data to be restored.

3.4.3.8 Operating system restoration

Restoration of an operating system's filesystem(s) will generally require booting from an alternate operating system such as those found on a rescue or Live CD so that operating system-specific files can be overwritten without crashing the system or resulting in data loss. Replacing existing system binaries and libraries while the system is active could result in system instability or inconsistencies. Other data such as configuration files can usually be restored while the system is active but only in so long as they are not actively opened for modification. Restoring device files will be depend on whether the device(s) are currently in use; if so, then the devices must be disabled or the system must be booted from an alternate device and then restored from media. It is very rare to ever have to restore device files from media as in most cases the Linux operating system can easily recreate them automatically at boot-up (using the same mechanisms responsible for operating system hardware reconfigurations).

In general, most operating system restorations can be done without ever having to reboot the system using an alternate device or brought into single-user mode as most of the operating system's files can be replaced without ever affecting the system. The only time this is necessary is when making changes to key system libraries, kernel files, and devices where system inconsistencies will likely cause a kernel panic or a total system crash rendering the operating system unusable or too unstable for use. In many cases, operating system backups can be restored to an alternate directory on the system and when applications and services can be disabled, the files can be replaced at a more appropriate time; this is the most suggested form for performing a restoration where the system's files cannot be immediately overwritten. Once all but the system's locked files have been overwritten or replaced, the system can be briefly rebooted and the locked files moved to their locations from the restoration made to alternate directory.

3.4.3.9 Users and applications

User-based data can generally be restored at any time; the only time this may be problematic is when a currently opened file is undergoing modification by the user and/or application or service. In such a case, the file cannot be replaced without causing data loss to the file. In such cases, it is normal for the system administrator to request that affected users be logged off the system by a given time so that the restoration can continue. In cases where the user cannot logoff from the system the data can always be restored to an alternate directory and then copied over to the appropriate location at a more appropriate time. If necessary, however, the system administrator can forcibly logoff a user and shut down his applications and open files; however, this should be left as a last resort to users in a state of non-compliance.

3.4.3.10 Databases

Database applications should always be shut down before restoring any database file. Unlike many other applications, database applications are data-specific and cannot generally be left operational even if the databases are currently unmounted; even unmounted database on quiescent systems can result in data loss, corruption, or database application crash if a database file is restored while the application is running. Database files do not necessarily require mounting in order to be considered as actively used by the application. In part, this can be attributed to the fact that database applications, generally initialized by scripts, may maintain journaling and buffered information about its files, even if they are unmounted. However, this will vary from application to application and as such, it is important to be somewhat familiar with the database application and how it works with respect to its database files. Reading the application's user and administrator manuals may provide further clarification.

3.4.3.11 File attributes

Not all backup tools are capable of storing extended file attributes. Depending on the tool used and the data backed up, the backup archive may or may not have saved the files' attributes. Furthermore, during restoration, it is important to use the correct restoration tool that is able extract and restore the various extended file attributes. The most common type of extended attribute is the ACL, and if incorrectly restored or backed up could result in the administrator having to manually recreating all the missing ACL's, a very time consuming process. *Tar* and *Cpio*-based backups do not store or restore extended file and filesystem attributes.

3.4.3.12 Security, compression, and networking

If security is a concern, then it is important that the same security measures be taken during restoration as was done during the backup. If certain applications such as *SSH* or *GPG* were used during the backup then they will also have to be used during the restoration in order to secure the network and/or data stream. Similarly, if the tools *SSH*, *RSH* or *Netcat* were used for the backup then they too should be used for the restoration. If encryption was used, then the passwords used must be available for data decryption. Furthermore, if administrative accounts are used for the backup, then the same administrative account will be required for a successful restoration. Access to system directories and direct disk access via raw devices also requires administrative access. Just as with data backups, certain permissions will be required in order to write to various parts of the system and to overwrite existing data. For most backup and restoration purposes, an administrative account will be required. In addition, various security mechanisms may be required in order to perform an over-the-network restoration; these types of restorations should generally be restricted to administrative accounts. The media should not be accessible to just anyone; only authorized personnel should be able to use and access backup media. Finally, if compression was used during the backup then the same tools will also be needed in order to decompress the network or data stream.

3.4.3.13 Multivolume restoration

Multivolume backups will require multivolume restorations. It is therefore important that any backup script(s) used also be capable of handling multivolume restorations otherwise user intervention will be required in order to perform these restorations. It is also a good notion to place these scripts onto any alternate boot devices so that they are readily available.

3.5 Miscellaneous

3.5.1 Errors

Errors can and always will occur at any time; sometimes predictably and other times randomly. Some errors are caused by human intervention (or lack of it) and others by software and/or hardware errors. An exhaustive list cannot be provided here since there are so many potential types of errors that can occur that a list could not possibly cover them all. Nevertheless, both the backup and restoration process should always be performed by someone knowledgeable of the computer system, the operating system, the applications and services used, the users, and the network layout – the system administrator. This individual is generally best suited to performing these vital tasks and functions.

In addition, the tools used for backing up and restoring data should be capable of dealing with and handling errors various types of errors. Unfortunately, most of them can only handle the most basic types of errors and therefore human intervention is usually required at one point or another when performing complex backups or restorations. Only very complex and costly backup tools can deal with errors that are more complex; however, no single tool could ever deal with all of the various problems that could potentially be encountered.

Errors can also occur in the backup media, not just during the actual phase of the backup or restoration. Media errors can be encountered when periodic media tests are conducted. The tools examined thus far can generally deal with bad blocks on the backup media. When these errors are found in the backup's archives, the tools can be used to make reasonable efforts to extract the data and if necessary skip to the next available backup file. In the worst case, *DD* can always be used to forcibly extract data from the backup media. However, portions of missing data from the archive(s) due to bad blocks may result in a partial or complete loss of data. Different tools will handle damaged archives differently. However, in general, all the tools examined are all able to deal with damaged archives and skip on to the next valid portion of the archive.

3.5.2 Testing

Before actually proceeding with any backup or restoration, it is important that both sets of procedures be adequately tested ahead of time in order to ascertain the applicability and efficacy of the procedures to adequately safeguard the data and the systems. In addition, the backup and restoration procedures should be tested and verified in order to determine that they do in fact work within the organization’s operational environments and conform to policy. The procedures should also be periodically retested to verify that they continue to conform to the operational environment and policy as changes can be made to the environment without the administrator’s knowledge, input, or consent, and as such, the procedures as they stand may no longer be suitable. Testing should always be done in a controlled laboratory or environment where changes can be made to the systems and network without affecting the operational network. However, once the tests and procedures have been tested and are considered ready for use, they should then be briefly tested on the operational network. The operational network test should consist of a short series of trial runs to determine if several of the most likely to be used procedures will work under varying conditions. However, this can only be done if the operational environment and organizational policy permit this and the affected systems can be taken offline, otherwise, laboratory-based testing will suffice. Once most or all of the tests are conclusive then the procedures can be rolled out into the operational environment.

3.6 Summary

As examined, performing data backups and restorations can be both a cumbersome and time-consuming process. There are many issues and sources of potential complication that must be analyzed and contended with before actually implementing and proceeding with any specific backup or restoration. After having read this section the reader should be ready to develop his own backup and restoration methodology. However, it is important to bear in mind that any methodology that is developed should always be specific to the organization’s requirements, policies, computer and operating systems.

In attempting to determine the necessary backup and restoration methodology, it is possible that much trial and error will be required in order to determine the best overall approach. Furthermore, until such time that a definitive backup and restoration policy and methodology are in place it may make more sense to use more than one type of backup tool and media until an appropriate combination is found that works and fits within the operational context as defined by the various requirements. In addition, the type of backup tool, the media to be used, as well as the type of data, and its size will largely determine the overall methodology and approach that will be used and implemented.

4. Methodology II – system maintenance steps and procedures

4.1 Objective

The objective of this section is to examine the various issues that surround operating system maintenance, whether it is for the update, upgrade, or manual system maintenance of an operating system. There are many valid reasons why operating system maintenance should be performed; for example, to fix bugs, improve performance, add newer functionality and improve system services, increase system stability, or prepare a system for an operating system hardware reconfiguration (or hardware migration). Fundamentally, this report's objective is to help prepare the system administrator with the necessary information required to adequately maintain computer systems so that they can be updated, either for its own sake or so they can undergo a hardware reconfiguration or migration (see reports [2, 3]) in order to accommodate for changes in hardware.

4.2 Introduction

Before any system maintenance-related action can be undertaken many issues will require a thorough examination by both the system administrator and support personnel who oversee the various C2 systems. Both this report and section take a more global overview of system maintenance as compared to Section 3 where precise actions and commands were examined in-depth. Throughout this section, a highly detailed discussion is not necessary as the proposed concepts are of a higher level and thus more vague and abstract. This in turns causes the reader to maintain a higher-level perspective that will help to apply the proposed methodology to various environments and organizational policies. In addition, it was determined that providing detailed commands and actions for this section could to lead to confusion instead of clarification. Linux distributions differ by varying amounts, thus attempting to write a lower-level system maintenance methodology that encompassed them all would be too complex and technically challenging to write in a Technical Note. However, there are enough similarities among them that a higher-level perspective could be examined.

Periodic and regular system maintenance is necessary to ensure that a computer operating system is both easier to maintain and administrate over the long-term. In addition, a well-maintained operating system will better adapt to periodic hardware changes facilitating reconfigurations and migrations. Furthermore, regular system maintenance, whether through regular updates, upgrades, code patching and/or manual recompilation/reinstallation all form a part of routine system maintenance necessary for the long-term smooth functioning of any operating system. The type of system maintenance that the reader will employ will largely depend on both the age and type of operating system in use.

This section provides a series of actionable overviews that can be taken up by system administrators and test users alike in order to develop a coherent update/upgrade methodology. Each specific organization is unique such that no simple cut-and-paste methodology can be applied at all times. Thus, this section focuses more on what should be done rather than how it is done. As such, detailed specific actions are removed from the discussion and left up to the discretion of qualified individuals who will carryout the various tasks. However, because of the approach taken throughout this section a higher level of UNIX expertise is required in order to perform many of the necessary lower-level system maintenance-related tasks. In addition, certain low-level tools (see Section 2.4.5.3) will be required for carrying out many of the system administration maintenance-related tasks.

Manual system maintenance, in contrast to updates and upgrades, should only be done on a case-by-case basis only when necessary. However, the level of expertise required for a thorough discussion of this topic is far beyond that found herein. By using the aforementioned tools low-level tool (see Section 2.4.5.3) many problematic issues found both for operating system updates, upgrades, and manual system maintenance can be resolved by tracking down the root cause of a problem (i.e. library inconsistency, incompatibility, etc.).

However, it is important to understand that at one point updates and upgrades may no longer be available (for a variety of reasons) leaving only manual system maintenance as the only method of maintaining an operating system. Although this is not an easy task, the discussions examined herein will be of immense when this scenario should occur.

4.3 System maintenance

4.3.1 Reasons for performing system maintenance

There are a variety of valid reasons why system maintenance should be performed regularly (or periodically), regardless of the maintenance type. The following is non-exhaustive list but it does cover many of the various issues that are important to this section. Consider the following:

a. Reduce the time required for future maintenance.

b. Decrease the system's susceptibility to attacks and exploitation through known vulnerabilities and/or bugs.

c. Potentially fix unknown vulnerabilities and/or bugs.

d. Ensure better hardware support for through newer device drivers and a newer kernel.

e. Improved management of both physical and virtual memory.

f. Provide increased system stability via newer device drivers, and provide enhanced hardware features and/or capabilities.

g. Through newer kernels provide improved system stability and reliability, new feature enhancements, increased security, and improved and increased hardware support.

h. Through updates and upgrades, provide newer system features and enhancements.

i. Enable a successful operating system hardware reconfiguration or hardware migration.

4.3.2 When and why to perform system maintenance

Before proceeding with system maintenance, in any of its various forms, it is important to determine why and if various operating system components should or need to be maintained. For example, if only one application or service requires maintenance then it may be more appropriate to manually modify the service/application instead of implementing an update or upgrade that will affect the entire system. Very often deciding on whether or not to carryout a specific course of action will require an impact assessment to determine what will change and whether the changes are justified vis-à-vis the amount of changes to be implemented.

Often times for small tasks manual system maintenance may be the preferable course of action. Certain system components can be easily maintained by manually while others should be done using updates and upgrades. For example, kernel recompilation using newer source code is not a difficult process, however, all too often required kernel features are excluded from the compilation that can potentially cripple the system. As such an update or upgrade may more appropriate as the kernel has already been configured for specific use with the current operating system. On the other hand, simple changes such as downloading and installing the latest office suite should cause no discernable impact on the system as a whole (other than new suite features). However, each case is different and will be dependent on many factors. Three very important factors to consider are: 1) are modifications required; 2) are they justified, and 3) if so through which means.

There are many reasons both in favour and against any particular course of action. It is therefore highly important to consider both the advantages and disadvantages of any specific course of action before proceeding with any system maintenance, even before laboratory testing. Doing otherwise could prove to be a waste of valuable time and resources that could otherwise have been better spent working on and solving other more pressing system administration-related issues.

There are many times when system maintenance should be performed and other times when it should not. The following is non-exhaustive list but does cover many of the various issues that are important to this section. Consider the following:

a. How difficult is the maintenance to implement? If the maintenance fix is complex or excessively long to implement then it should be done using a more automated method such as an update or upgrade.

b. What requires changing? If manual installation of the maintenance fix will cause too many potential changes, incompatibilities, or inconsistencies then they should be implemented using an automated method such as an update or upgrade.

c. What requires maintenance? Does the operating system in general require maintenance or specific components such as the kernel (and its subsystems) or system services and applications? Updates are often more effective at providing newer kernels and system components as they are already precompiled for the current platform and distribution.

d. Is maintenance to be performed against one or more applications and/or libraries? If this is the case then it is often easier to perform manual system maintenance in so long as the source code does not need to be modified in any major way.

e. Does the maintenance require source code modification? If too much source code is to be modified then time might be saved by instead implementing an update or upgrade. On the other hand if there is only a small amount of source to modify then possibly manual system maintenance may be the correct choice of action.

f. Is the maintenance necessary to fix one or more specific system bugs? System bugs such as library or kernel bugs should generally be fixed using updates or upgrades rather than manually modifying kernel source code in order not to introduce new incompatibilities or inconsistencies. However, this will depend on the type of bugs and their breadth.

g. If the bugs are application-related, do they cause usability issues? It is important to consider if it is worth updating or upgrading an application simply to fix a couple of bugs that cause only minor inconveniences. Sometimes newer applications cause more trouble than they are worth (i.e. user retraining). It is therefore important to determine the severity of the bugs before implementing any changes.

h. Will performing maintenance provide any new productivity improvements (i.e. improved functionality or performance, or both) in applications and services or will it improve the overall operating system? What type of changes will be made? Are they far reaching or limited in their scope? Answering these questions will help to determine if the changes should be carried out.

i. Is maintenance required to fix various security-related concerns? Depending on the security issue at hand, it may not be necessary to correct, as it may only be applicable to unused system services/applications. On the other hand, not fixing highly used services/applications could increase the system's overall vulnerability.

j. Are new vulnerability and/or other security maintenance fixes available? If they can be implemented without excessively changing or affecting the system then they can be implemented using manual system maintenance. If on the other hand the maintenance fixes are complex to implement or make too many changes (known and unknown) to the system then it may be more appropriate to use an automated method such as an update or upgrade.

k. Are the security-related maintenance fixes kernel-related? If so, then it is often best to directly apply the fix to the source code and recompile the kernel. This can also apply to binary patches. However, if the level of changes made to the source code is extensive then it may more appropriate to use an automated method such as an update or upgrade. In addition, kernel recompilation and option selection can be complex and can result in a crippled kernel resulting in an unusable system.

l. What are the overall advantages and disadvantages of performing system maintenance? If the advantages outweigh the disadvantages then the system should be maintained using the appropriate maintenance type. However, if there are too many disadvantages using one maintenance type then another type should be used in its place.

m. It is important to consider when the last maintenance-related changes were made. If the system is relatively up to date then it may not be necessary or appropriate to make changes to the system for non-critical maintenance fixes. Sometimes making maintenance-related changes may cause far more work than is necessary for the level of maintenance to be provided.

n. Are the maintenance changes critical? If so then they should be implemented unless there is a good reason not to (i.e. services and/or applications rendered non-functional). However, if maintenance-related changes are not implemented immediately then when a more appropriate set is available such as update or upgrade and it does not cause excessive adverse affects then it should be implemented.

4.3.3 Requirements for system maintenance

Before proceeding with any system maintenance, there are certain requirements that should be met. No system changes should have been made until these requirements are ascertained. Doing otherwise may result in an operable system that would have to be restored from backup media. The following is non-exhaustive list but does cover many of the various issues that are important to this section. Consider the following:

a. What requires maintenance? Does the operating system require it or do various components, applications, libraries, and/or other packages require it? Prepare a list detailing specifically requires maintenance. If enough components require maintenance then it is likely a good decision to proceed with system maintenance.

b. Determine which type of maintenance is to occur. If only a small number of changes or modifications must be made to the system then manual system maintenance may be more appropriate. If many changes are to occur then an update should occur. If updates are no longer available or certain new technical innovations or fixes are available in the upgrade then it may be necessary to perform an upgrade.

c. Are all license issues resolved? Has the operating system been re-licensed (this has the potential to happen) as updating or upgrading to it would cause the current operating system to fall under the new licenses jurisdiction.

d. If manual system maintenance is to be applied then it is important to determine if any changes have been made to the software licenses and their corresponding software packages. Software packages modified by the maintainer's updates or upgrades are the maintainer's legal responsibility. However, software packages that are manually maintained whose underlying license has changed remains the legal problem of the customer.

e. Is the test laboratory environment ready for use? Is the test computer's hardware and software installed and working? Before changes are ever deployed to the operational network they should always be tested and verified on a non-operational network that can be experimented on.

f. Have system diagnostics been run on the hardware? Diagnostics should be run prior to performing an update or upgrade as a hardware bug could cause the entire process to come to a halt and even corrupt the operating system. In addition, certain hardware configuration checks should be made in order to ascertain that the system is in good working order including verifying the system's log files for any hardware-related issues.

g. It is important to verify the system's logs and determine that there are no outstanding errors or bugs unless the purpose of maintenance is to fix those specific issues.

h. Has the software and operating systems on the test hardware been tested to determine if they are functional? Is the operating system in a functional state? A system in proper working condition is easier to maintain than an unstable system.

i. Have backups been successfully completed? It is important that backups be available as nay failed attempt at performing system maintenance could require the reloading of system data from backup media. It is important to ensure that backups are done periodically for both the laboratory testing facility and the operational network.

j. Are the test users ready for testing the system after system maintenance is performed? If system maintenance will have a visible impact on the users or the way they work it will important to test what changes they will experience and whether the changes are more conducive or a hindrance. If the changes are more of a hindrance then the changes should be rolled back.

k. Is the system administrator(s) ready to proceed with the system maintenance? Does he have everything he requires? It is important that the system administrator and his support staff be ready to proceed with the system maintenance modifications and that all materials (i.e. update/upgrade media, backups, etc.) are available should they be required.

l. Is the operating system ready to receive an update, upgrade, or manual system maintenance? Are applications and services disabled? Have users logged off from the system? It is common practice before performing system maintenance that any application or service that may be affected be disabled and that users who would be affected be logged off from the system. In addition, it may be necessary to bring the system to single-user mode or even boot from special media if performing an update or upgrade.

The following is non-exhaustive list but it does cover many of the various issues that are important to this section.

4.4 Maintenance types

Although this section places emphasis on the long-term system maintenance of Linux-based computer operating systems, other types of system maintenance are equally suitable depending on the current period from the original deployment of the C2 system. In general, while the various forms of system maintenance achieve the same overall objective, they are accomplished through different means (i.e. updates, upgrades, code patching, etc.). These various forms of maintenance can be broken down into three general categories: short-term, medium-term, and long-term system maintenance. These three forms of system maintenance can be accomplished using any or all of the various maintenance options provided in [2], although there is likely to be some overlap.

These three maintenance types should help to satisfy the Navy's key requirement: that a computer operating system is maintainable for the duration of the C2 system's lifetime. This lifetime is likely to be a period of at least 15 years and possibly as long as 25 years. Currently, it is still uncertain which computer operating system will replace the current C2 operating system aboard the Halifax-class frigates; however, when they are refitted it is very likely that a Linux-based operating system will power the new systems.

4.4.1 Short-term

Short-term maintenance is generally the easiest and simplest type to perform on a given Linux operating system. The operating system is generally maintained using updates provided by the distribution's maintainer. Commercially supported Linux operating systems can easily expect updates to be available for at least the first 2 to 3 years of the system's life. However, different support contracts with a given vendor may allow for extended periods of operating system updates for possibly as long as 5 years. Updates are normally provided for the duration of the support period of the operating system in so long as the distribution's maintainer has not switched to a newer version and has ceased support for older operating systems.

Most commercial distributions are able to directly download and install updates, as they become available using sophisticated GUI-based applications. Support contracts may also stipulate that instead updates be to be provided on physical such as quarterly update CD's.

Updates do not generally cause large operating system disturbances. Instead, they usually provide bug fixes and software patches, but can also provide improved application and system reliability and performance, and less commonly provide improved application functionality. Generally, most updates are transparent to the end users, although from time to time it can occur that certain applications may change slightly from update to another. Normally, however, this should pose no problem to the end users or to system configuration files. In addition, currently supported applications and libraries should not be affected by maintainer-based updates, as they should already have been thoroughly tested by the maintainer. Only rarely do updates upset the balance of system libraries, although when it does happen manual system maintenance may be required to rectify certain inevitable problems. These problems can often be fixed by creating new configuration environments[4] for the affected applications and libraries.

Updates periodically include newer kernels, although drastic changes in kernels (i.e. switching from kernel 2.4.x to 2.6.x) are indeed a rare occurrence. Minor kernel changes do not normally

[4] A configuration environment could be one of several things. Normally it is when an application and its libraries are moved to another location on the system and a new system and/or user configuration is created for it and is independent of the rest of the system. Another type is creating a *chroot*-based environment where the application runs inside of another more restricted environment. Unfortunately, these environments are complex to setup and configure and may not always be an appropriate solution.

result in changes to system calls and which therefore have no tangible impact on the system's existing applications and libraries. Nevertheless, keeping the operating system's kernel up to date will improve a system's ability to undergo a hardware reconfiguration or migration.

Finally, manual system maintenance can be used at any time to manually update and upgrade any application, library, service, or kernel. However, due to the amount of additional work required when performing manual system maintenance related tasks its use is likely to be rather limited, although it is always available if it serves as a more viable solution to an update.

4.4.2 Medium-term

As operating system update support is dropped in favour of supporting more recent versions of the operating system, an upgrade will eventually be required to move up to the next level of technology unless for some reason manual system maintenance is preferred. Upgrades, however, are usually preferable to manually maintaining and supporting an operating system due to the complexity involved. Thus, an upgrade is an operating system medium-term maintenance solution. It can be expected that a deployed operating system will likely have to undergo an upgrade from one operating system version to a newer one around every 5 years, unless extended operating system update support is stipulated in the support contract with the vendor.

However, simply because update support has stopped may not be incentive enough to justify upgrading, especially if large changes are entailed. Conversely, upgrading may be justifiable due to new technical innovations by providing features such as a substantially newer kernel, applications, services, and libraries. An older operating system can normally be upgraded with only a few problems to be expected along the way. It is, however, probable that at least several important applications may no longer work as expected due to application and/or library-based changes. If this occurs, part of the upgrade can normally be rolled back and the missing files recovered from backup. Then a new configuration environment can be created for the affected applications and libraries. This may not always fix the problem, however, and sometimes the only way to fix it is to track down the root cause(s) of the problem using the tools detailed in Section 2.4.5.3. In the worst of cases, applications and/or libraries that do not work properly can be recompiled and/or modified from source code; however, this can be time consuming and complex to the uninitiated.

Therefore, unless there are valid reasons for not upgrading to a newer operating system version when update support has been terminated, in-house laboratory testing should be conducted to confirm that upgrading would result in a stable operating system. In addition, in-house testing will confirm if the changes caused by an upgrade conform to organizational requirements and policy (i.e. changes in operating system security policies).

Once the system and its applications have been deemed functional, the system can then be updated using vendor provided updates. Once an upgrade is successful, subsequent updates can then be applied to the system until the next major upgrade is available or necessary. It is likely that updates for commercially supported operating systems will be available for as long as another 5 years before the next major upgrade is required.

Therefore, upgrading is a medium to long-term system maintenance solution. Its caveat is that upgrades eventually must themselves be superseded by a series of updates only to be followed by another upgrade. With time, through the successive implementation of updates and upgrades the operating system may begin to show symptoms of software decay. Software decay occurs when enough small changes have been made to a computer system where it no longer behaves as expected due to the number of progressive modifications that have substantially altered the system, its usability and reliability. This is generally caused by an excess of library incompatibilities and inconsistencies that can no longer be easily managed through alternate configuration environments. Although rare this can sometimes be caused by system call changes in the kernel. This type of issue is almost never seen for updates designed for the original operating system but can and may occur as time progresses and subsequent upgrades and updates have been applied to the system. This may take as long as 15 or more years before coming into full effect.

4.4.3 Long-term

The final type of maintenance to examine is manual system maintenance. This form of maintenance is generally suitable for either long-term system maintenance (i.e. may occur after 10 to 15 years of continuous maintenance) or when one or more of several possible events has occurred. The most likely events to occur are:

1) The distribution's maintainer has gone out of business.

2) The maintainer has been bought out by another company that either does not support or sell Linux-based products.

3) The maintainer no longer supports Linux as it has changed its line of business.

4) Alternatively, if an organization can no longer benefit from the use of successive upgrades and updates as too of them has rendered key applications and libraries non-functional.

On the hand, manual system maintenance may be required as a permanent solution after many upgrades and updates as the time required to fix the problems caused by them is equal to or greater than the amount of time necessary to manually maintain the system.

Manual system maintenance can be used at any time during an operating system's lifecycle in order to continue supporting and maintaining it and its applications and libraries; however, this is not advisable as it only be used when necessary. It is almost never used after the initial deployment of the C2 system as updates can generally fulfill its role. It is only after many years and too many system changes that it should be seriously considered. However, this is not to say that every software package must be manually recompiled and reinstalled. Rather, manual system maintenance can be the manual and direct application of updates and upgrades without using any GUI installation program. Instead, the system administrator becomes responsible for ensuring that required software packages are up to date through whichever means are at his disposal. Equally applicable is the implementation of new or modified source code, recompiling and reinstalling it. In so doing, the system administrator gains additional control over which applications and libraries are changed.

What sets long-term manual system maintenance apart from the other types is that the system maintenance is now almost entirely dependent on the system administrator, the method he chooses (as the situation requires) and his skills and expertise in maintaining the system. In addition, it is very likely that the system administrator will not be able to carryout all the required tasks by himself and will have to work in with other technical support staff. However, in so long as the applications and libraries in use continue to be supported by the vendor manual system maintenance may only required occasionally. Fortunately manual system maintenance is only necessary when applications and/or libraries are no longer functional due to various sources of conflict.

Manual maintenance is unfortunately the longest and most complex form of system maintenance possible and generally requires extensive in-house testing. Sometimes it will require the direct modification of existing source code and at other times the application of patches or simple downloads and installations. Cases will vary widely and many situations are likely to be unique and equally cumbersome and complex to implement. Although long-term manual system maintenance does not necessarily need to be used by the Navy so long as other maintenance options remain open and available to it [2]. It is entirely possible that through highly specific support contracts customized updates and upgrades can be provided from the vendor that will be able to adequately satisfy the Navy's long-term requirements. However, as is the case with business things change and without at least giving this option due consideration the Navy may find itself without any other viable option other than to deploy a completely new operating system from scratch instead of building upon and reusing available resources.

4.5 Licensing

Software licensing can be a contentious issue and it can have a dramatic impact on which type of long-term maintenance to use. Almost all FOSS-based software is distributed with a specific type of software license. There are many different types of licenses and although many of them are similar, each one grants specific rights and limitations to the end-user. Commonly the end-user is considered the organization that is using or deploying the software in question. However, this definition equally applies individuals using the same software at home.

It is common for software licensing to become a complicated and convoluted matter not just for government departments but also for any involved party. The problem with software licenses is that the language they employ is often subtle, vague, and all too often all encompassing. Unfortunately, where software licenses are concerned there are often many subtle nuances that have to be understood in order for the end-user to understand his rights and limitations. However, licenses tend to grant few rights although they do generally contain many legal limitations. Report [3] may serve as an interesting starting point for those interested in comparing two different commercial licenses used for FOSS. In the report, a table has been provided to provide a useful comparative examination.

However, FOSS licenses tend to be far less restrictive than their commercial counterparts are. Commercial Linux distributions, while open source in nature, are generally bundled with commercial licenses that are often more restrictive than most FOSS-based licenses. However, these licenses tend not to be as confining as other non-FOSS commercial licenses.

4.5.1 Types

An important question to ask before deploying any open source product is what type of license is the software bundled under? There are currently more than 50 different types of open source licenses, and while many are similar, they each have their own specific advantages and drawbacks, rights and limitations. The two most commonly encountered are GPL or BSD-based. Many of the other currently available licenses are derivations of these two types.

In a military setting where intellectual property (IP) is important it is advised that a BSD-based license be used as it allows the end-user to preserve full rights to any changes he makes to the source code. Conversely, GPL-based licenses generally require that all changes made to the source code be given back to the community. Where matters of national security are concerned this may not be in the best interest of Canada.

It is therefore important to understand the implication of the license type for the C2 system and what types of maintenance can be performed on it. If maintenance is provided by the vendor [2] then all legal responsibility rests with them, otherwise if maintenance is to be done in-house then a legal analysis of the license should be considered before adopting it. Unfortunately, the Canadian government has not yet reached a decision about how to treat open source licenses (this is normally the jurisdiction of Justice Canada).

4.5.2 Compatibility

License compatibility issues are important but are often overlooked, whether for FOSS or proprietary software. An important question to consider is ‘does the software license allow me to make modifications to the operating system?’ In addition, as software changes over time so to do the underlying licenses. It is common for FOSS software to change to another license type. This gives rise to the following question: ‘how will changes in FOSS licenses affect subsequent modification of that software?’ License changes occur in large part do to pressure from the software’s users and developers, but sometimes it occurs if the software maintainer has a philosophical “change of heart.” While this is very unlikely to occur for well-established FOSS projects such as the Linux kernel, core system tools and utilities the possibility always exists. License changes may also be necessary to ward off claims of intellectual property [19].

If system maintenance is to be required in the near future, it is important to determine if current software licenses are the same of compatible with their newer counterparts. The answer will depend on many factors the most important of which is the governing license that all other software licenses fall under. For example, a commercially based distribution has a governing license that states that its license supersedes all other underlying licenses. Thus, all changes to the software and underlying licenses are the responsibility of the vendor, not the end user. Therefore, if a software package changes its license to another type that is incompatible with the current governing license it remains the vendor’s legal problem.

Conversely, those performing manual system maintenance and are no longer have a valid support contract with the vendor fall outside the protective umbrella of the vendor’s governing software license. Thus should underlying software licenses change from one type to another then legal counsel should be sought out before proceeding to make any changes. All too often, changes to FOSS licenses, although minor in nature, will allow the end user to continue working with and modifying the source code. However, in the event that the new license is incompatible with the needs of the end user (i.e. divulging intellectual property) he can, at his discretion choose to replace the software package for another type that satisfies his specific requirements. The other alternative is to not adopt the newer software package with its incompatible license and manually maintain the older software without using any source code from the newer software or from the user community. For example, a FOSS package originally developed under the BSD license is eventually re-licensed under the GPL license. This would severely affect new source code modifications and require them to be distributed back to the community. The only way around it either use and maintain the older source code or switch for another similar FOSS software package.

4.5.3 Permissions and limitations

It is important to understand what is permitted under both a governing commercial software license and the various packages' underlying licenses. Each license grants a specific set of permissions and limitations. Some licenses, commercial and open source, are very restrictive while others are far more permissive. For example, GPL-based licenses are far more restrictive than their BSD counterparts are. However, much will depend on what is specifically required of the license and of the software, including any potential changes to be made to the source code, both in the present and future. For example, if a software component's license has changed and it no longer reflects the needs of the organization it may be more appropriate to replace it with another component whose license is more in line with the requirements of the organization. Thus, the permissions and limitations sought after in a license should be closely reflected by the organization's requirements of the software and of the C2 system. It is also important to consider that a governing licenses change over time and the rights and limitations given to the end user also change as per updates and upgrades implemented against the various systems. Therefore, accepting newer versions of software can have a direct impact on the type of maintenance to use which can affect the C2 system's long-term evolution.

4.5.4 For consideration

As with all legal issues, operating system and software licenses vary from case to and from package to package and distribution to distribution. The vendor may simply apply a standard generic license to the distribution thus covering all the underlying software with its license. Some of the issues that have not yet been examined are as follows:

a. If the distribution is bundled using a standard commercial generic license, is manual system maintenance permissible? If it is permissible, what are its limitations? If not, then legal counsel should be sought out before attempting manual system maintenance.

b. Depending on the vendor's commercial license it may be necessary to renew on a yearly basis the software support contract simply to have the right to use the distribution, let alone modify it [3].

c. Do the open source licenses found with the distribution allow for the modification of the kernel, system configurations, applications and libraries? Generally, this is not an issue; however, this may be problematic if source code and IP are integrated.

d. Have any of the software packages' licenses changed? If they have, are they compatible with existing licenses and with both current and future modifications to source code? What is the impact of these license changes?

e. Can the vendor's distribution source code be patched or integrated with non-vendor source code that has been provided by the open source community or third-party?

f. According to the governing license can specific packages be changed, modified, or replaced? What do the licenses of the individual packages permit?

4.6 Laboratory testing

Before considering performing any type of system maintenance-related action on an operational C2 computer system, whether it is an update, upgrade, or manual system maintenance, all actions should first be thoroughly tested. Specifically, tests should be conducted within the confines of a special laboratory environment where tests can be safely performed and evaluated without jeopardizing the stability of the operational network. A laboratory is an exceptional setting for testing patches, bug fixes, updates, upgrades, source code modifications, etc., before they are ever rolled out and made operational. Laboratory testing provides an opportune time and location to perform system reaccreditation and recertification in order to minimize the overall impact that would otherwise be experienced in operational environments. Since the Navy has expressly stated that all changes must be certified and tested, this setting provides a unique opportunity for carrying out these necessities. Once a test system can be reaccredited and recertified in a laboratory setting it can then be safely implemented (barring certain precautionary safeguards) onto operational systems.

In addition, those interested in software degradation and operating system evolution will appreciate the use of such facilities to study and examine how operating systems and their software respond to many changes progressively made over the years. At the same time, some may see laboratory testing as a waste of time, especially in testing small and seemingly innocuous changes. Nevertheless, even small changes made progressively overtime build up and can cause a "snowball effect" which can effectively cause seemingly functional systems to cease functioning.

4.6.1 Laboratory

Before proceeding with any laboratory tests, it is important that the testing environment be as similar as possible to the operational environment. The laboratory does not need to have the same number of computers or users. However, the laboratory should utilize the same type of telecommunication equipment used in the operational network such as routers, bridges, uplinks, etc. The physical computer systems should be of the same make and model and those in the operational network as software can respond differently when used on different systems. This is in fact a well-known problem of computer system, particularly hardware dependent software such as kernels and device drivers. In addition, when troubleshooting this will help to isolate software related problems caused by incompatibilities and inconsistencies from hardware issues. Furthermore, the operating systems, applications, services, etc. should be similarly configured as the systems they are meant to represent on the operational network. It is very important that the systems be as similar as possible in order for laboratory testing to be meaningful and produce useful results.

In addition, similar environments will help to "shake down" test systems and reveal software and hardware bugs before they are encountered in the operational network. Furthermore, a laboratory will help to provide a more realistic environment for system administrators, support staff, and users to test and experiment with the various systems. This will help to determine if applications, databases, files, telecommunication systems, and other various services are equally available, responsive, and functional as they were before any system modifications were made. In the end, all this testing will help to enable a faster and more simplified deployment and transition of the required changes (i.e. updates, etc.) onto an operational setting.

4.6.2 Laboratory isolation

It is important before proceeding with any tests or modifications that the test environment (i.e. laboratory) be completely isolated from any of the operational networks. Both networks need to be free from any potential source of contamination that could be caused the other. If either environment were interconnected it would make bug tracking and troubleshooting more problematic. Even environments separated by NAT-based firewall systems cannot stop all possible cross-contamination issues. Thus, when conducting tests on C2 systems and their networks it is important to determine the sources of possible outside influence in order to track down bugs or other issues. Furthermore, the extra layer of objectivity afforded by isolation will make system reaccredidation and recertification longer and more complex as outside sources may have to be taken into account.

4.6.3 Backing up

It most certainly is important to backup all data before proceeding with software testing in the event changes must be rolled back. However, depending on the specifics of the test environment, the software used and installed, backups may not always be necessary, although they are often a good idea. In addition to preserving data backups will allow for the additional testing of new backup technologies, methodologies, and emergency restoration procedures.

Data to be backed up will vary considerably and will be according to the type of data and test systems to be backed up. A backup may consist of user data, applications, and configurations or it could be a full system backup. However, the backing up multiple test systems may result in an excessive work and may therefore be more appropriate to use cloning and distribution systems (i.e. Norton *Ghost*) instead. This approach is particularly well suited to environments where all or most of the systems are identical. More information on backing up can be found in Section 3.

4.6.4 Benchmarking

Before deploying any successful changes made to the C2 test systems onto the operational network, it is important to consider the performance-based issues that can inadvertently affect operating systems. One way to determine if a set of "successful" modifications will cause inadvertent changes on the operational network is to test them in some way. The standard method of testing consists of examining application and service-based functionality as well as usability tests. Another method that is proposed herein is benchmarking.

Benchmarking is a performance-based test that measures the performance of the system, application, or service against some measurable unit. The most commonly used unit is time, although it any other useful performance-based unit is permissible. Benchmarking is useful because it can help to pinpoint slowdowns caused by new software or a set of modifications. It verifies if performance (could be the system as a whole or an application or service) is similar to before implemented changes. By benchmarking the system when important modifications are made to it various side effects, adverse or otherwise can be made known. Modifications can be tested individually or in groupings of likeminded changes. Furthermore, it possible to benchmark software according to changes in both the software itself and their configurations.

By benchmarking different modifications and attributing a performance-based score to pre- and post-modification systems it is possible to determine with good accuracy whether a given change or set of changes has been successful. A general rule of thumb is proposed: if the performance of a system, application, etc., is closely similar to the original then the change(s) can be considered successful. Conversely, noticeable slowdowns can be indicative of problems requiring resolution or simply of an inadequate or incompatible change or set of changes having been made.

The theory behind benchmarking is that non-effectual or counterproductive changes and modifications are more likely to cause system, application, and service slowdowns. Thus, if a set of modifications actually causes an unexpected and significant slowdown where none was previously seen then it is likely that those changes are either incorrectly set or are detrimental to the system and should be reversed.

However, in order for benchmarks to be useful, a base score is required and this can only be accomplished by benchmarking the original test systems C2 systems marked for deployment. The system, as well as key applications and services should be benchmarked and serve as a comparison for future benchmarks. In addition, benchmark results can vary widely due to extraneous factors; thus benchmarks should be repeated several times in order to average out the result. In so doing, benchmarking can help to objectively pinpoint potential performance gains and problems that are the result of one or more modifications.

4.6.5 Incremental changes

It is important when testing systems in a laboratory that changes, wherever possible, be made incrementally. This is not always practical or possible, however, to put into practice. When testing various maintainer-provided updates and upgrades, depending on how they are to be implemented, incremental implementation may not be possible although with tweaking it often is. Certainly, this is easier to achieve when using manual system maintenance as compared to maintainer-based updates and upgrades as these generally require an installation program that does not always make it possible to make incrementally changes. However, most distributions provide the ability to fine tune system and application updating; upgrading on the other hand is often far more problematic. Nevertheless, this will vary considerably from distribution to distribution and from version to version. At the same time, configuration files can also be changed incrementally, making a set of changes and testing them before proceeding with another set of changes. These changes can be small or large and consist of one or more configuration files.

Making changes incrementally is always a best practice; however, time does not often permit for this. After each modification (or set of) it is appropriate to perform benchmarking tests and comparisons. Incremental changes can allow for several goals can be attained:

1) It enables a more precise targeting and tracking of problems, instabilities, and inconsistencies that arise because of changes and/or modifications.

2) Facilitates the rolling back of changes, as there is less to be removed and undone as compared to a full system update or upgrade.

3) It makes version and system change control easier to track and maintain.

Unfortunately, during this process, much will depend on the distribution itself as different operating systems use various approaches to applying updates and upgrades. In general, upgrades tend to make incremental testing difficult and sometimes for all practical purposes infeasible.

4.6.6 System administration testing

It is obvious that the system administrator will know the various systems, infrastructures, telecommunication equipment, and operating systems very well. It is his job to understand them and to be comfortable with them. That is why, although obvious to state, that the system administrator plays a key role in testing the system after changes have been made to it. The system administrator above all others knows what to expect and how the system in general should behave, including its performance, reliability, security configurations, and network-based access and resources. Of course, user testing is also very important. Before user-based testing is done system administration testing should take precedence. Only after the system administrator finds the system to be functional should other tests be conducted on it. Different system administrators, according to the skills and years of experience way use determine a system's suitability for work using different tools and techniques. While most vendor distributions provide similar UNIX-based tools they can sometimes behave differently. Thorough documentation of system administrator-based testing is as important to note as any other test. In addition, results from the various tools including system metric information should also be included in any documentation.

4.6.7 Behaviour and functionality

Before accepting a set of modifications there are several issues that are very important to consider. The most important of these is to notice if behavioural are a result of the modifications made to it. Behavioural changes could be indicated by a change in screens or system messages when the system boots up or powers down. There could also be various messages written to the console that could be the result of one or more buggy device drivers. Applications and/or services that were once fast and responsive are now slower or unresponsive (i.e. benchmarking). Noticing behavioural changes often is not an easy task, but someone such as the system administrator should be familiar enough with the current hardware and operating systems to recognize many types of differences. Many things can be changed after an update or upgrade and this is why, when possible, changes should be made incrementally, tested, and observed.

The system administrator must determine if the functionality of the operating system is essentially the same as it was before the change(s) were made. It is only normal that updates and upgrades will change the kernel and other key operating system components, but these changes should not adversely affect the system. However, the system administrator is uniquely qualified to determine if aberrant behaviour or functionality is a results of changes made to the system.

However, the system administrator is not likely to recognize changes made to the various applications used on the system(s) by the various users. Different users will use different applications and services to start and complete their assigned work and tasks. Test users are uniquely capable of determining whether applications and services are behaving and functioning correctly. Test users should consist of advanced users who are fully capable of performing their tasks with the least amount of system administrative support so that they can independently verify if adverse changes are present.

Benchmarking can be used to help alert the system administrator to various problems by attempting to determine how performance after a set of system modifications differs from the baseline benchmark(s). It is also important to determine if there are any noticeable or adverse changes or to the system after the implementation of specific software or configuration file changes and/or modifications. Of course, aberrant system, application, or service behaviour is not always caused by direct binary modification; sometimes it is caused by changes to configuration files. Other times it is caused by a change to one or more library interdependencies or system call changes.

4.6.8 User-related system changes

User-related system changes must be evaluated in order to determine whether any of the changes made will cause system, application, or service-related disruption or failure. Any type of disruption or failure could adversely affect the way users work and interact with both the system and each other. This will in turn adversely affect both their day-to-day tasks and their overall mission objectives, many of which have to be accomplished together as a team. In a mission critical environment, any disruption could be potentially disastrous.

Therefore, it is very important to test the system after a set of changes, even if they are small. Although the system administrator is generally able to distinguish adverse affects to the operating system itself, only the users are uniquely positioned to test the system's applications and services. Of course, not every change is necessarily a bad; in fact, most of often changes are necessary so to fix bugs and provide newer or enhanced features.

Laboratory-based testing in an isolated environment will make it easier to determine if the users experience any differences in their day-to-day activities and use of the system before changes are eventually deployed onto operational systems. This provides the users an opportunity to voice themselves beforehand rather than be forced to accept non-functional or aberrant modifications or changes to their applications and work methodologies. Thus, user-based impact studies are necessary in order to assess the usability of the changes made. Some tests users should consist of "power users," users who are generally self-sufficient. System administrators in general make poor test users as they often bypass security or organizational procedures to accomplish their task(s) or fail to understand how application-based changes will affect users.

User-based changes should be tested on a case-by-case and not all changes require test users. Operating system changes that are sure not to affect applications and the users do not necessarily require user testing (of course, the system administrator should test these changes).

4.6.9 Impact assessment

For every set of changes made to the system an impact assessment should be conducted; however, the implementation of an impact assessment should be commensurate with the amount time necessary to actually perform one as well as the time required to make the changes. For example, if several very small changes are made and the consequence(s) of these changes are already well known ahead of time then it may not be necessary to carryout an impact assessment. Judgement and common sense should be utilized at all times; otherwise, the impact assessment portion of testing could become excessively complex and cumbersome.

Before conducting any impact assessment two questions should be asked. The first is “will the system(s) and network(s) continue to behave as they always have after the changes are made?” The second question applies more specifically to system behaviour and is “if a set of changes makes no visible changes to the system or the users’ ability to interact with it then it is worth performing an impact assessment?”

Impact assessment-related issues are important to determine. Unfortunately, not all changes and their impacts can be known or understood ahead of time; however, many are known as they are already well documented and may even have been previously tested at a different time or place. Nevertheless, impact assessment should be conducted from within a laboratory-based environment. In addition, this laboratory is critical in order to adequately perform and test patches, updates, upgrades, manual system maintenance and bug fixes in order to look for potential incompatibilities, incoherencies, or new instabilities that could be introduced into the operational environment or infrastructure. These issues can be caused by changing and/or replacing key operating system libraries and interdependencies, applications, system and application configurations as well as configurations. Furthermore, it is important to determine if any of the changes break or modify system and organizational security policies. Impact assessment testing should be used in conjunction with benchmarking and system administrator and user-based testing.

4.6.10 Modifying system configurations

It is important to consider the impact of system configuration modifications. Some changes are innocuous and are likely to result in no noticeable changes or behaviour to the system; others may cause services and applications to act different from before. For example, the upgrade of an important network service was configured via its configuration file to block certain types of connections from external certain systems. Now that it has been upgraded and a new configuration file has replaced the previous one these blockages are no longer in effect. This may seem a minor detail (depending on what is affected) but this could potentially allow unauthorized access to data or services that are ordinarily unavailable. In addition, the modification of system configurations can dramatically affect the users their interaction with the system. For example, users connect using a network-based protocol but after a series of changes a service configuration is modified and now compatibility mode for older clients is deactivated.

There are certain issues that should therefore be examined during the different testing phases after an update or upgrade has been implemented. It is important to determine which files have been changed, particularly configuration files. Changed configuration files should be compared to their previous incarnations to determine if any important service or system-wide changes have been made or propagated. This is another reason why it is also important to conduct backups prior to implementing updates and/or upgrades. Furthermore, this is why documentation, change assessment, and versioning control are important.

4.6.11 Outcome testing

It is important to determine if a set of changes made to the system results in a desirable outcome; specifically, to determine whether the changes were successful or a failure. However, it is important to define beforehand what should be considered a success and a failure. Failure could be defined as the causality between a specific set of changes and a disruption of services or applications, system stability, or reliability. It could also be defined as an unacceptable change in system behaviour, performance, or application and/or service use. Only after a thorough examination and stringent testing can the cause of a problem or failure, if it occurred, be determined. Then, depending on its manageability it can deemed a success or failure. This, however, generally requires testing on the parts of both the system administrator and test users. Both have a complex job ahead of them; however, it is often more difficult for the users to determine whether their applications and work methodologies continue to work appropriately. This can include their ability to establish access and work with data and other repositories, system services and applications, as well as accessing and using remote systems and devices (i.e. printers, etc.). In addition, if things are not working as they should they must also determine where they fall short.

On the other hand, if the changes and the overall impact on the system are considered acceptable in that they have caused little to no discernable problems or disruptions then the outcome for the changes can be considered a success. The case is further solidified if a set of changes imparts additional benefits such as improvements in usability, performance, security, robustness, etc., without adversely affecting the system.

Outcome testing is not actual testing phase per se. Instead it is a culmination of results obtained from benchmarking, impact analysis, user tests, system administration related tests, etc. All successes and failures should be thoroughly documented as well as justification for an apparent success or failure. It is to be expected after an update, but especially following an upgrade that there will be some failures. However, in so long as they remain manageable then there is no need to consider the entire update or upgrade a failure. Manageable failures can still be included in the update or upgrade process while those that cannot be reasonably managed should either be altogether left out or fixed, if possible, using manual system maintenance.

4.6.12 Versioning and change control

Versioning control is rather simple to carryout in so long as the proper preparations have been made (i.e. versioning control software has been installed). Many software packages are available that can perform versioning control for various UNIX and Linux-based operating systems. This software allows for the analysis and determination of which files have been changed, by whom, and when; some can even go as far as comparing changes against archived copies. This information is important in order to assess which process has made changes (i.e. update or upgrade) and the nature of the change.

However, before versioning control can be implemented, a baseline must be established. Almost all (if not all) versioning control software requires a baseline to be established. A baseline can be used to establish information about a system's files such as size, ownership, permissions, creation date, access date, modification date, etc. It is generally possible to specify which files or sets of files should be taken into account while creating the baseline (i.e. specific configuration files, directories, binaries and libraries, etc). With this information it becomes possible to determine which files have changed.

Versioning control information is generally stored in a database file; this file tends to text-based. The baseline data file should always be considered as an important starting point for any documentation that is to be written up. In addition, through thorough versioning control and documentation it will be possible to maintain an established list of known changes that can be used to help track down software and configuration errors as a troubleshooting aid.

4.6.13 Library and kernel modifications

Most updates and upgrades will affect multiple libraries as well as the kernel. Depending on the type of system maintenance utilized, system changes may be minor or widespread. Generally, library and kernel changes tend to provide additional functionality, improved security, and feature and bug fixes. It is generally rare that required features such as API's and system calls will be removed, although it is always a possibility. When time and resources permit, it is always best to ascertain the specifics to changes in these files. Using a versioning control system it will possible to isolate changed files from unscathed ones with relative ease (assuming the files have been baselined). While it is not often necessary to examine these files in-depth, if problems or other issues should arise as a direct result of an update or upgrade, then the changed files will have to be examined.

Changes to system calls are rather easy to determine if kernel source code is readily available, otherwise specific tools will be required to extract this information. Libraries on the other hand are generally more difficult and time-intensive to analyze, although there are tools directly designed for this purpose. It may be appropriate to analyze libraries only if as a direct result of their modification one or more applications or services malfunctions or fails. A full listing of these tools is available in Section 2.4.5.3.

However, determining when if it is worthwhile to proceed with an analysis is of great importance, as is the specific use of which tools to use, particularly if resources are scarce. Unfortunately, large library-based changes (more common with upgrades than updates) is considerably more difficult investigate thoroughly because of the far-reaching changes have been imparted. This is particularly true for critical libraries such as the *libc* library (C library) which provides most of the operating system's and applications' C calls and functionality. Discovering which applications and services would have been affected by a change to a core library is a cumbersome and time consuming. Therefore, it is important to decide if and under which conditions this analysis is to be conducted.

4.6.14 Reconfiguration and migration

Once all of the tests have been conducted and it has been determined that the overall outcome thus far has been successful, then if appropriate, it is time to proceed with an operating system hardware reconfiguration or hardware migration. This step should only be conducted if new hardware has been introduced onto one or more of the test systems. For systems that have not experienced a change in hardware then this step should be altogether skipped. In certain circumstances where the kernel and/or libraries have not been changed a reconfiguration or migration can still be done in so long as the kernel supports the newer hardware or that a third-party device driver be available.

Unfortunately, due to the proliferation of many diverse incarnations of the Linux operating system there is no generic approach to performing a reconfiguration or migration. Most modern Linux distributions have their own particular method for detecting hardware changes and making the appropriate operating system changes and configuration file changes. This topic has been thoroughly examined in Report [1].

4.6.15 Documentation

The importance of documentation cannot be overstated. It is important that at least one individual, preferably the system administrator (or other similar person) document information about the various changes and tests carried out. Documentation should be written in a clear and understandable language that is objective. The documentation should include but not be limited to versioning and change control information, changes and other information relevant to library, application, and system interdependencies. It is also important to include listings of changed files and packages, configuration file modifications, as well as kernel and driver changes. Equally important are the various tests that have been performed: impact and outcome, benchmarking, behaviour testing, and system administrator and user-based tests.

The documentation should be able to convey to any technically qualified person all the required and necessary information about the changes experienced by a system including the various results obtained from benchmarking, behaviour and user-related tests, as well as performance evaluations. The information collected for documentation purposes will be vital to system reaccredidation and recertification as every system change, modification, and overall impact will already have been conducted and detailed. Therefore, if the changes made by an update or upgrade are successful and are thoroughly documented then deployment of approved changes will be more seamless. Documentation will of course be a requirement in order to gain approval for deploying a set of changes onto operational systems.

Documentation should also consist of problem resolution and other successful troubleshooting techniques that managed to resolve problems caused by an update or upgrade. These problems are likely to occur again when the changes are deployed onto operational systems; thus, without this information the process will be both more cumbersome and time consuming.

All results, whether good or bad, should be documented. A case should then be made and available in the documentation detailing the reasons why an update or upgrade should be allowed to proceed. An objective analysis of all tests, changes, and documentation will facilitate the approval process. It is likely that many organizations will have their own procedures and policies for writing technical documentation and approval must have been given be proceeding with any deployment.

4.6.16 Approval process

At this point, once all tests have been appropriately conducted according to requirements, time constraints, available testing resources, documented, and troubleshooted (if necessary) an overall outcome should be apparent. Regardless if the outcome is positive or negative a case should be made why a set of changes (i.e. update/upgrade) should or should not be deployed. Using the objectively written documentation as well as overall assessment put together by technical personnel management can make an informed decision about whether or not to proceed with a deployment. The importance and quality of the documentation provided to management cannot be overstated, as its decision will be largely based on the conclusions and findings found within the documentation. Once approval has been given to deploy, a deployment plan should be developed and a course of action for reaccredidation and recertification put into place.

4.7 Deployment

Once all required tests have been conducted, documented, and approved the update and/or upgrade can be deployed onto operational systems and networks. However, there is much planning that remains to be done in order to determine which systems the modifications are to be deployed onto and the order of operation and priority. It will also be important to coordinate deployment efforts with appropriate IT personnel so that they are available for deploying the approved changes as well as troubleshooting if necessary. Other issues such as reaccreditation and recertification are examined in this section.

4.7.1 Backing up

Prior to deploying an approved changes onto operational systems and networks full backups should be conducted so that if necessary system states can be rolled back if one or more system deployments should go amiss. Although the approved changes have been thoroughly tested in a controlled laboratory environment the possibility exists that some type of failure could occur during the deployment. Such a failure could result in network-wide disruptions potentially leading to the entire unavailability of services and capabilities. Therefore, by backing up any system that may be affected by the deployment it will be possible to restore them to their original state. A backup and restoration methodology can be found in Section 3 of this report.

4.7.2 Deployment plan

It is important to develop a deployment plan. The plan will examine many issues that must be resolved in order to appropriately plan and deploy the various updates and/or upgrades. Not doing so leaves many complex variables to chance and can considerably increase the probability that a deployment effort will go amiss. A non-exhaustive list of potentially contentious issues to consider has been provided below:

a. How many systems will the update, upgrade or manual system maintenance are deployed? Is the deployment to be broken into small or large groups or done across many systems simultaneously?

b. Are there enough available resources to proceed with the deployment?

c. Does the organization have a policy in place for deployments? Does the proposed deployment coincide with existing policy or other frameworks?

d. What will be the effects on the other systems that will not be directly involved in the deployment and will not receive a given set of modifications? This should have been tested in a lab setting, documented, and approved.

e. Will the operational network or infrastructure be destabilized by the deployment? This issue should have been tested in a lab. It may be necessary to temporarily disconnect the network so not affect other systems, networks, and services.

f. Will the deployment be performed during time allotted for routine system maintenance (weekends, holidays, etc.)?

g. Will deployments be done one system at time, in groups, or all at the same time? Each option will require its own specific planning.

h. How will deployments be done? Will update or upgrade-based deployments be done in part, starting with the kernel, then libraries, services, and finishing with applications to minimize user downtime and impact? Alternatively, will the deployments be done by implementing the complete update and/or upgrade at the same time? These scenarios should have been tested in a lab setting.

i. Can users continue to work during the deployment? Will they be able to access their data and applications and perform their routine tasks? This should have been tested in a lab setting with the test users.

Once the deployment plan has been developed, it should be approved by both management and reaccredidation and recertification officials. The modifications as laid out in the deployment plan are made in the following section.

4.7.3 Rollout

The deployment should proceed according to the deployment plan. The deployment plan should be developed in terms resource availability as well as organizational policy and procedure. It is here that the actual modifications to operational systems are made.

Due to extensive laboratory testing, few unknown problems and other issues should be encountered during the operational deployment although the possibility of this occurring exists. This is because laboratory testing cannot take everything into account that is to be found in operational settings. It is conceivable that a deployment could have interacted in unforeseen ways with the operational network. These inevitabilities cannot be taken into as they tend to be random in nature and can potentially caused by many difficult to pinpoint sources of origin (i.e. electrical short-circuit, cosmic rays, solar flare, etc.).

Regardless of the potential for failure (which should be relatively low), the deployment should follow as closely as possible the development plan put together in the previous section. The rollout phase should be documented just as has been done for all of the previous steps because should failure(s) occur then using well-written documentation it may be possible to track the root cause of the problem or failure.

4.7.4 Reconfiguration and migration

As stated in Section 4.6.14, a reconfiguration or migration is necessary only if hardware changes have been made to one or more of the underlying systems present on the operational network or infrastructure. If no hardware changes have been made then neither a reconfiguration nor migration is necessary. However, if changes have been made then in order for that new hardware to function correctly the operating system will have to recognize it. However, each distribution is unique and each operating system will have its own mechanism for detecting hardware changes and make them available to the operating system. It is beyond the scope of this report to directly examine or detail the specifics concerning reconfiguration and migration; more information can be found in Report [1].

4.7.5 Reaccredidation and recertification

If all the previous steps have been appropriately completed and the changes have been deemed successful, the documentation is objective, accurate, up to date, and provides extensive coverage of events and testing then reaccredidation and recertification should be a rapid process. While the process varies according to organizational policy, the ultimate goal of the process is to determine what has changed and what its impact will be on the system and network. By performing all of the aforementioned steps, these time-consuming verifications are taken out of the loop of IT security personnel and left in the hands of those better able to determine the cause and effect implications of the various system changes that have been made. In addition, once all appropriate evidence has been collected and corroborated from previous steps there should be a high degree of certainty about overall system reliability and stability. The systems should then be deemed judged satisfactory for operational use and given final approval by reaccredidation and recertification officials.

4.7.6 Wrap-up

Once the deployment has been successfully completed, reaccredited and recertified it will require a “shakedown” period where hidden bugs not found during laboratory testing or deployment can be worked out. A successful shakedown could take several weeks to several months to complete and to learn about any new undocumented features about the changes implemented. During this time, it is important to document any lessons learned (if any) and examine any last minute changes, tweaks, or modifications that have to be made to accommodate for the effectuated modifications and changes.

5. Conclusion

There is no clear-cut or definitive methodology for carrying out system backups or performing system maintenance. The purpose of this Technical Note has been to propose two methodologies to aid system administrators in these tasks. Different operating systems will require different methodologies. However, because the Navy is interested in deploying Linux-based systems as their new C2 systems aboard the retrofitted Halifax-class frigates only Linux has been examined. The material presented herein is applicable, in general, to both Linux and UNIX-based operating systems.

The first methodology, found in Section 3 examines the various techniques and technical issues surrounding performing quality system backups before testing and deployment of system maintenance. Different maintenance types and their consequences have been examined in Section 4. Here, a system maintenance methodology has been developed to aid in the testing and deployment of various types of system maintenance.

The Navy has expressed their interest in maintaining the same operating system throughout the lifecycle of the C2 system. As such, they will inevitably have to perform maintenance on these operating systems, and when they do, the issues examined herein will be of great use to both their system administrators and other technical personnel whose job it is to provide support and maintenance. Many may not agree with the contents herein, however, this is quite likely the first document of its kind as no other system maintenance methodology could be found for Linux or UNIX in general.

In conclusion, although Linux distributions vary greatly according to their market niches, they all share certain features and similarities. It is based on these similarities and features that the methodologies proposed in sections 3 and 4 have been proposed. Of course, they are open to interpretation and can be changed to suit different requirements and environments, but at their basis, they offer relevant and useful tips and advice. Section 2 provides useful background information concerning system maintenance, operating systems, and the various types of dependencies likely to be encountered.

References

[1] Carbone, Richard. Operating system hardware reconfiguration: A case study for Linux. Technical Memorandum. Defence R&D Canada. TM 2006-595. November 2006. http://cradpdf.drdc.gc.ca/PDFS/unc56/p527008.pdf.

[2] Charpentier, Robert, and Carbone, Richard. Life-Cycle Support for Information Systems Based on Free and Open Source Software. Revision 1.0. Technical Paper for 11th ICCRTS. Defence R&D Canada. June 2006. http://www.dodccrp.org/11th_ICCRTS/abstracts/136.pdf.

[3] Carbone, Richard. Enterprise Linux licenses: A comparison of licenses between Red Hat and Suse Enterprise Linux. Technical Note. Defence R&D Canada. TN 2006-573. October 2006. http://cradpdf.drdc.gc.ca/PDFS/unc53/p526349.pdf.

[4] Wikipedia. System call. Online encyclopaedia. Wikimedia Foundation Inc. October 2006. http://en.wikipedia.org/wiki/System_call.

[5] Carbone, Richard. Does Red Hat 5.0 Support Hardware Refreshes and can it Work on Modern x86 CPU's. Internal Report. Defence R&D Canada. November 2005.

[6] Carbone, Richard. Can Linux be Easily Reconfigured from One Machine to the Next?" Internal Report. Defence R&D Canada. October 2005.

[7] Carbone, Richard. A What to do Avoid Guide in Doing your own In-House Migration. Internal Report. Defence R&D Canada. November 2005.

[8] Michaud, Frederic, and Carbone, Richard. Practical verification and safeguard tools for C/C++. Technical Memorandum. Defence R&D Canada. Document No. TR 2006-735.

[9] Painchaud, Frederic and Carbone, Richard. Java software verification tools: Evaluation and recommended methodology. Technical Memorandum. Defence R&D Canada. Document No. TM 2005-226. March 2006. http://cradpdf.drdc.gc.ca/PDFS/unc57/p527369.pdf.

[10] Weimer, Hendrik. Dissecting Programs. Online article. OS Reviews. September 2006. http://www.osreviews.net/reviews/admin/strace.

[11] Ravi. strace – A very powerful troubleshooting tool for all Linux users. Online article. All about Linux. May 2006. http://linuxhelp.blogspot.com/2006/05/strace-very-powerful-troubleshooting.html.

[12] Cespedes, Juan. Ltrace Linux man page. Man page. Die.net. http://www.die.net/doc/linux/man/man1/ltrace.1.html.

[13] Maurer, Ben. Memory usage with Smaps. Online article. Ben Maurer. March 2006. http://bmaurer.blogspot.com/2006/03/memory-usage-with-smaps.html.

[14] Nguyen, Binh. Linux Filesystem Hierachry. Revision 0.65. Howto guide. The Linux Documentation Project. July 2004. http://tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html.

[15] Burford, Sean. Introduction to Reverse Engineering Software in Linux. Revision 1.26. Howto guide. University of Adelaide. September 2002. http://www.ouah.org/RevEng/t1.htm.

[16] Die.net. ldd man page. Man page. Die.net. http://www.computerhope.com/unix/uldd.htm.

[17] Free Software Foundation. GNU Binary Utilities. Guide. Free Software Foundation. May 2002. http://www.gnu.org/software/binutils/manual/html_mono/binutils.html.

[18] Abell, Victor A. Lsof man page. Revision 4.63. Man page. NetAdminTools.com. http://www.netadmintools.com/html/lsof.man.html.

[19] McDougall, Paul. Microsoft Claims Linux Infringes 42 Patents. Online article. Information Week. May 2007. http://www.informationweek.com/news/showArticle.jhtml?articleID=199501578.

List of symbols/abbreviations/acronyms/initialisms

ACL	Access Control List
API	Application Programming Interface
BSD	Berkeley Software Distribution
C2	Command and Control
CD	Compact Disc
CRC	Cyclic Redundancy Check
DD	Data Definition
DMSS	Directorate of Maritime Ship Support
DRDC	Defence Research Development Canada
DVD	Digital Video Disc
Ext2/Ext3	Second Extended Filesystem / Third Extended Filesystem
FOSS	Free and Open Source Software
FSCK	Filesystem Check / File System Consistency Checker
GNU	GNU Not UNIX
GPG	GNU Privacy Guard
GUI	Graphical User Interface
HMCCS	Halifax Modernized Command Control System
I/O	Input/Output
IDE	Integrated Device Electronics
IT	Information Technology
NAT	Network Address Translation
PKI	Public Key Infrastructure
R&D	Research & Development
RAM	Random Access Memory
RHEL	Red Hat Enterprise Linux
RSH	Remote Shell
SCSI	Small Computer System Interface
SSH	Secure Shell
Tar	Tape Archiver
U.S.	United States

This page intentionally left blank.

Distribution list

Document No.: DRDC Valcartier TN 2007-150

LIST PART 1: Internal Distribution by Centre:

3	Document Library
1	Richard Carbone (author)
1	Robert Charpentier
1	Michel Lizotte
1	Guy Turcotte
1	Mario Couture
8	TOTAL LIST PART 1

LIST PART 2: External Distribution by DRDKIM

DRDC Corporate HQ

1	LCol Peter Scott (DSTC4ISR 3)
1	Donna Wood (DSTC4ISR 4)
1	Directorate R & D – Knowledge and Information Management

NDHQ (101 Colonel By, Ottawa, K1A 0K2)

1	Directorate of Maritime Support Systems Attn: Mr. Norbert Haché
1	Directorate Information Management Strategic Planning 3 Attn: Maj. J.Perry Mellway
1	Directorate Information Management Strategic Planning 2-2 Attn: Maj. P.C. Kvas
1	Directorate of Aerospace Engineering Support 5 Attn: Sylvain Fleurant
1	Directorate Information Management Requirements 4-6 Attn: Mark Daniels
1	Directorate Distributed Computing Engineering and Integration 3-5 Attn: Capt. Karine Pellerin
1	Directorate Information Management Security 2-3-2-2 Attn: Paul Lamoureux
10	TOTAL LIST PART 2

18 TOTAL COPIES REQUIRED

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.) DRDC Valcartier	2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.) Unclassified

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C, R or U) in parentheses after the title.)

(U) Long-term operating system maintenance: A Linux case study

4. AUTHORS (last name, followed by initials – ranks, titles, etc. not to be used)

Carbone, R.

5. DATE OF PUBLICATION (Month and year of publication of document.) January 2008	6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.) 67	6b. NO. OF REFS (Total cited in document.) 19

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Note

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

DMSS HMCCS

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.) 1430JT 15AV34	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.) DRDC Valcartier TN 2007-150	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

(X) Unlimited distribution
(   ) Defence departments and defence contractors; further distribution only as approved
(   ) Defence departments and Canadian defence contractors; further distribution only as approved
(   ) Government departments and agencies; further distribution only as approved
(   ) Defence departments; further distribution only as approved
(   ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

In *Operating system hardware reconfiguration: A case study for Linux*, it was determined through experimentation that a Linux-based C2 operating system can successfully undergo a hardware migration and operating system hardware reconfiguration. The direct benefit of this is the ability to forgo any new operating system reinstallation in order to support newer hardware by using mechanisms internal to the operating system that support changes in hardware; this results in a decreased waiting time for system reaccredidation and redeployment. Since an operating system can evolve over time, it can accommodate changes in the system's hardware, thus presenting a tangible advantage for the Navy as this allows the operating system to be maintained over the long-term. However, there are complexities involved when maintaining an operating system for long periods. Therefore, this report serves as an introduction and a simple methodology for performing system maintenance-related tasks that include upgrading, updating, as well as data backups and restoration. This report is neither all-inclusive nor a replacement for qualified system administrators with years of experience. Instead, it can be used as a useful source of information to provide recommended practices, procedures, and information to help in planning for long-term system maintenance.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

free and open source
FOSS
hardware migration
hardware reconfiguration
kernel
Linux
migration
operating system
operating system hardware reconfiguration
operating system reconfiguration
patching
reconfiguration
system administration
system maintenance
upgrade
update